

普通高等教育“十一五”国家级规划教材  
21 世纪大学计算机规划教材

# 计算机硬件技术基础

## （第2版）

何 桥 主编  
段清明 藏雪柏 副主编  
赵宏伟 马爱民 刘 威 梁 燕 何 威 编

電子工業出版社  
**Publishing House of Electronics Industry**  
北京·BEIJING

## 内 容 简 介

本书为普通高等教育“十一五”国家级规划教材。

本书系统地介绍了微型计算机的硬件技术及应用基础。主要内容包括：微型计算机硬件基础知识、微处理器（CPU）及其系统机构、指令系统和汇编语言程序设计、总线技术、存储器、输入/输出系统、中断系统、定时/计数器、接口电路、A/D 和 D/A、外部设备及其接口、单片计算机应用技术等，形成了一个完整的、系统的计算机硬件技术基础教学内容。

本书在内容的取舍上尽量做到少而精，力图通俗易懂，由浅入深，通过实例和习题加深对基本概念的理解和掌握。在每章之后均配有习题，供自学自测用。

本书可作为高等学校非计算机专业本科、专科的计算机硬件技术基础、微机原理及应用、微机接口技术、单片计算机应用技术课程的教学用书，也可作为高等学校成人教育的培训教材和教学参考书，还可以供从事微机应用开发工作的科技人员参考。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

## 图书在版编目（CIP）数据

计算机硬件技术基础 / 何桥主编. —2 版. —北京：电子工业出版社，2009.7

（21 世纪大学计算机规划教材）

ISBN 978-7-121-09058-5

I. 计… II. 何… III. 硬件—高等学校—教材 IV. TP303

中国版本图书馆 CIP 数据核字（2009）第 096808 号

责任编辑：韩同平 特约编辑：刘汝辉

印 刷：

装 订：

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编：100036

开 本：787×1092 1/16 印张：21.5 字数：550 千字

印 次：2009 年 7 月第 1 次印刷

印 数：3000 册 定价：29.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 [zlts@phei.com.cn](mailto:zlts@phei.com.cn)，盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

服务热线：（010）88258888。

## 第 2 版前言

本书为普通高等教育“十一五”国家级规划教材。

随着计算机技术的飞速发展和计算机应用的日益普及，对高等学校非计算机专业的计算机教育提出了越来越高的要求。根据普通高等教育“十一五”国家级规划教材编写的要求，为适应新技术发展，满足教学要求，这次对第 1 版教材做了修订，力争使修订后的教材能反映计算机硬件技术的新知识、新技术、新方法，努力与计算机应用技术发展同步，形成完整的、系统的微型计算机硬件技术基础教学内容。

鉴于非计算机专业所涉及的专业较多，不同专业之间教学内容差别很大，本书在编写时遵循了非计算机专业的特点，采用模块化、结构化的内容组织原则，以具有较宽的适用面和灵活的选择余地，利于实施不同层次、不同对象的教学。在内容的取舍上尽量做到少而精，力图通俗易懂，由浅入深，通过实例和习题加深对基本概念的理解和掌握。

本书内容选材上注重科学性和实用性，基础知识和实际应用结合比较好，教学内容适应计算机应用发展需要。在原教材的基础上增加了一些新内容、新信息，增加了微型计算机硬件技术应用实例。教材内容的组织注重微型计算机硬件技术基础和应用能力培养。

本书第 1、3、6 章由何桥编写，第 5、7、8、9 章由段清明编写，第 2、4、10 章由臧雪柏编写，6.4 节、7.7 节由赵宏伟编写，第 11 章由马爱民、刘威、何威编写。全书由何桥统稿。本书电子课件由梁燕制作。

由于作者水平和经验有限，难免有不足之处，敬请广大同仁和读者批评指正。

作者的电子邮件地址：[he-qiao@sohu.com](mailto:he-qiao@sohu.com)

编 者

# 目 录

第 1 章 计算机系统概述	(1)
1.1 计算机发展概述	(2)
1.2 微型计算机的分类	(4)
1.3 计算机的应用领域	(4)
1.4 计算机硬件基础	(5)
1.4.1 计算机中数的表示和运算	(5)
1.4.2 微型计算机的基本组成电路	(13)
1.5 微型计算机系统	(19)
1.5.1 微型计算机系统的组成	(19)
1.5.2 微型计算机的基本结构	(21)
1.6 微处理器的组成	(21)
1.7 微型计算机系统的主要性能指标	(22)
1.8 微型计算机的一般工作过程	(23)
习题	(24)
第 2 章 80x86 微处理器及其系统结构	(25)
2.1 8086/8088 的内部结构	(26)
2.1.1 8086/8088 的编程结构	(26)
2.1.2 8086/8088 的寄存器结构	(27)
2.1.3 8086/8088 的存储器组织及地址形成	(29)
2.1.4 8086/8088 的 I/O 端口组织	(31)
2.2 8086/8088 的外部结构	(31)
2.3 8088 的工作模式	(35)
2.4 8086/8088 的总线操作和时序	(36)
2.5 8086/8088 的横向提升	(38)
2.5.1 数值数据协处理器 8087	(38)
2.5.2 输入/输出协处理器 8089	(39)
2.6 80x86 高档微处理器	(40)
2.6.1 80286 的体系结构	(40)
2.6.2 80386 的体系结构	(43)
2.6.3 80486 的体系结构	(46)
2.6.4 Pentium 微处理器的体系结构	(47)
习题	(49)
第 3 章 指令系统及汇编语言程序设计	(50)

---

3.1	寻址方式 .....	(51)
3.2	指令系统 .....	(52)
3.2.1	数据传送指令 .....	(53)
3.2.2	算术运算指令 .....	(56)
3.2.3	逻辑运算和移位指令 .....	(60)
3.2.4	串操作指令 .....	(63)
3.2.5	输入/输出指令 .....	(66)
3.2.6	控制转移指令 .....	(66)
3.2.7	处理器控制指令 .....	(69)
3.3	系统功能调用 .....	(70)
3.4	汇编语言程序设计 .....	(71)
3.4.1	汇编语言的语言格式 .....	(72)
3.4.2	常数 .....	(73)
3.4.3	伪指令 .....	(73)
3.4.4	汇编语言源程序的结构 .....	(76)
3.4.5	汇编语言程序举例 .....	(77)
	习题 .....	(85)
<b>第4章</b>	<b>总线技术 .....</b>	<b>(87)</b>
4.1	总线的基本概念 .....	(88)
4.2	IBM PC 总线 .....	(92)
4.3	ISA 总线 .....	(94)
4.4	PCI 总线 .....	(94)
4.4.1	PCI 总线的特点 .....	(95)
4.4.2	PCI 总线的系统结构 .....	(96)
4.4.3	PCI 总线信号 .....	(97)
4.5	STD 总线 .....	(100)
4.6	主要外设总线介绍 .....	(102)
4.6.1	USB 总线 .....	(102)
4.6.2	IDE 总线 .....	(104)
4.6.3	SCSI 总线 .....	(106)
4.6.4	IEEE 1394 总线 .....	(107)
4.6.5	AGP 总线 .....	(109)
4.6.6	IEEE 488 总线 .....	(110)
4.6.7	CAN 总线 .....	(112)
	习题 .....	(113)
<b>第5章</b>	<b>存储器 .....</b>	<b>(115)</b>
5.1	半导体存储器概述 .....	(116)
5.1.1	半导体存储器的分类 .....	(116)

---

5.1.2 半导体存储器的结构	(117)
5.1.3 半导体存储器的主要性能指标	(118)
5.2 半导体存储器芯片	(118)
5.2.1 半导体存储器与 CPU 总线相关的信号线	(118)
5.2.2 半导体存储器芯片的外特性	(120)
5.3 半导体存储器的应用	(124)
5.3.1 半导体存储器电路的分析方法	(124)
5.3.2 半导体存储器在计算机系统中的应用	(128)
习题	(130)
<b>第 6 章 输入/输出系统</b>	<b>(131)</b>
6.1 接口概念	(132)
6.2 CPU 与 I/O 设备之间的接口信息	(132)
6.3 CPU 与外设之间的数据传送方式	(133)
6.3.1 无条件传送方式	(133)
6.3.2 查询传送方式	(134)
6.3.3 中断传送方式	(136)
6.3.4 直接存储器存取 (DMA) 控制方式	(136)
6.4 DMA 控制器 8237A	(137)
6.4.1 8237A 的内部结构	(138)
6.4.2 8237A 的引脚功能	(141)
6.4.3 8237A 的工作方式	(142)
6.4.4 8237A 的编程	(145)
习题	(148)
<b>第 7 章 中断系统</b>	<b>(149)</b>
7.1 中断概述	(150)
7.1.1 中断的必要性	(150)
7.1.2 中断源	(151)
7.1.3 中断系统的功能	(151)
7.2 CPU 响应中断的条件和过程	(152)
7.2.1 CPU 响应中断的条件	(152)
7.2.2 CPU 对中断的响应	(153)
7.3 中断优先权及多重中断	(154)
7.3.1 中断优先权	(154)
7.3.2 多级中断的概念	(157)
7.4 8088 的中断方式	(158)
7.5 IBM PC/XT 的中断方式	(161)
7.6 中断控制器 8259A	(163)
7.6.1 8259A 的内部结构	(164)

7.6.2	8259A 的引脚功能 .....	(165)
7.6.3	8259A 的工作方式 .....	(167)
7.6.4	8259A 的编程 .....	(171)
习题	.....	(176)
第 8 章	可编程定时/计数器 8253 .....	(177)
8.1	概述 .....	(178)
8.1.1	8253 的内部结构 .....	(178)
8.1.2	8253 的引脚功能 .....	(179)
8.1.3	8253 的控制字 .....	(180)
8.1.4	8253 的工作方式 .....	(181)
8.2	8253 的编程 .....	(185)
习题	.....	(186)
第 9 章	接口电路 .....	(187)
9.1	可编程并行接口 8255A .....	(188)
9.1.1	8255A 的结构 .....	(188)
9.1.2	8255A 的工作方式 .....	(189)
9.2	可编程多功能接口 8155 .....	(195)
9.2.1	8155 的结构及引脚 .....	(195)
9.2.2	8155 的工作方式与基本操作 .....	(196)
9.3	串行通信及可编程异步通信接口 8250 .....	(199)
9.3.1	串行通信基础 .....	(199)
9.3.2	8250 的内部结构 .....	(202)
9.3.3	8250 的引脚功能 .....	(204)
9.3.4	8250 的编程 .....	(206)
9.4	D/A 转换及其接口 .....	(209)
9.4.1	D/A 转换原理 .....	(209)
9.4.2	8 位 D/A 转换器 .....	(210)
9.4.3	8 位 CPU 与超过 8 位的 DAC 接口 .....	(213)
9.4.4	12 位 D/A 转换器 .....	(214)
9.5	A/D 转换及其接口 .....	(215)
9.5.1	A/D 转换的基本过程及转换原理 .....	(215)
9.5.2	8 位 A/D 转换器 .....	(220)
9.5.3	12 位 A/D 转换器 .....	(225)
9.5.4	双积分式 A/D 转换器 .....	(230)
9.6	闭环控制系统 .....	(231)
习题	.....	(232)
第 10 章	外部设备及其接口 .....	(233)
10.1	概述 .....	(234)

10.2	键盘及其接口	(234)
10.2.1	消除抖动及重键处理	(234)
10.2.2	线性键盘	(236)
10.2.3	矩阵键盘	(237)
10.2.4	键盘工作方式	(242)
10.2.5	PC 键盘与接口	(242)
10.2.6	BIOS 键盘中断及 DOS 键盘功能调用	(245)
10.3	LED 显示器及其接口	(248)
10.3.1	七段 LED 显示器结构	(248)
10.3.2	LED 显示器的显示方式	(249)
10.3.3	LED 显示器接口及应用举例	(250)
10.4	打印机及其接口	(254)
10.5	视频系统	(257)
10.5.1	CRT 显示器	(257)
10.5.2	液晶显示器	(258)
10.5.3	字符和图形显示的基本原理	(261)
10.5.4	显示器的主要性能指标	(265)
10.5.5	显示适配器	(267)
10.5.6	对显示器的编程	(272)
10.6	鼠标器及其接口	(276)
10.7	其他外部设备	(277)
10.7.1	扫描仪	(277)
10.7.2	绘图仪	(279)
	习题	(280)
<b>第 11 章</b>	<b>MCS-51 单片机</b>	<b>(282)</b>
11.1	MCS-51 单片机的组成	(283)
11.2	MCS-51 单片机的芯片引脚	(284)
11.3	存储器配置	(285)
11.4	时钟电路及时序	(288)
11.5	定时/计数器	(290)
11.6	中断控制系统	(293)
11.6.1	中断系统结构	(293)
11.6.2	中断系统的控制	(295)
11.7	串行口	(296)
11.8	MCS-51 单片机指令系统	(298)
11.8.1	寻址方式	(298)
11.8.2	指令格式及说明	(299)
11.8.3	数据传送类指令	(300)



11.8.4 算术运算类指令 .....	(302)
11.8.5 逻辑运算指令 .....	(304)
11.8.6 位操作指令 .....	(305)
11.8.7 控制转移类指令 .....	(306)
11.9 应用举例 .....	(308)
习题 .....	(313)
附录 A 8088/8086 运算指令对标志位的影响 .....	(315)
附录 B DOS 功能调用 (INT 21H) .....	(316)
附录 C BIOS 功能调用 .....	(322)
附录 D MCS-51 指令表 .....	(327)
参考文献 .....	(332)

# 第 1 章

## 计算机系统概述

### 教学目的和要求

本章主要介绍计算机的发展及应用领域、微机分类、计算机硬件技术基础知识、微机系统的组成、微机一般工作过程。要求读者了解计算机的发展及应用领域，重点掌握计算机硬件技术基础知识、微机的组成、结构特点，为后面各章的学习奠定基础。

## 1.1 计算机发展概述

电子计算机诞生于 20 世纪 40 年代，它的出现是 20 世纪的重大科学技术成就之一，有力地推动了各门科学技术的发展，其应用已深入到科学文化、工农业生产、国防建设甚至于家庭厨房等各个领域，成为科学研究、工农业生产和社会生活所不可缺少的重要设备。计算机的应用程度成了衡量一个国家现代化的重要尺度。

在推动计算机发展的诸多因素中，电子元器件的发展是一个重要因素。电子计算机更新换代的主要标志，除了电子元器件的更新之外，还有计算机系统结构方面的改进和计算机软件发展等重要内涵。计算机更新换代的大体时间划分如下：

第一代（1946~1958 年），电子管计算机。在美国，为了解决军事上的需要，由美国宾夕法尼亚大学设计的数字电子计算机 ENIAC（Electronic Numerical Intergrator And Calculator）于 1946 年诞生。这台世界上的第一台计算机，是一个庞然大物。它有 18 800 多个电子管，1500 个继电器，重达 30 吨，占地 170 平方米，耗电 150 千瓦，价值 48 万多美元，运算速度 5000 次/秒，与今天的微型计算机相比真是不可同日而语了。但是，它却奠定了电子计算机的技术基础，如采用二进制数进行运算和控制，建立程序设计的概念等。

第二代（1958~1964 年），晶体管计算机。这一代计算机的硬件部分采用了晶体管，主存储器采用铁氧磁心和磁鼓，外存储器已开始使用磁盘，软件已开始有很大的发展，还提出了操作系统的概念，出现了各种高级语言及编译程序。这一代计算机除进行科学计算之外，在数据处理方面也有广泛的应用。

第三代（1964~1971 年），集成电路计算机。随着半导体集成技术的发展，使得几十、几百甚至上千个元件能够集成在只有几平方毫米的半导体芯片上。这一代计算机采用中、小规模集成电路取代了晶体管分立元件。采用集成电路后，计算机的体积进一步缩小，耗电减少，可靠性和运行速度明显提高。在技术上引进了多道程序和并行处理，操作系统的功能也不断加强和趋于完善，这些都更加方便了人们对计算机的使用。在这一时期，计算机在科学计算、数据处理和过程控制等方面都得到了较广泛的应用。

第四代（1971 年至今），大规模集成电路计算机。这一代计算机的元器件采用了大规模集成电路，软件更加丰富，数据库系统迅速普及并开始形成网络，操作系统的功能更为强大，图像识别、语音处理和多媒体技术有了很大发展。

计算机更新换代的显著特点是体积缩小，重量减轻，速度提高，成本降低，可靠性增强。微型计算机是我们目前接触最多的计算机。正是由于微型计算机的发展与普及，才使计算机应用范围迅速拓展到目前社会活动的几乎所有领域。微型计算机系统升级换代的标志有两个，一个是微处理器，另一个是系统组成。微处理器的发展主要表现为字长的增加和速度的提高。

1971 年 Intel 公司研制成功 4 位微处理器 4004，时钟频率 740kHz，它应用于各类袖珍计算器进行简单运算，或者用于家用电器和娱乐器件中进行简单的过程控制。

1973 年推出了 Intel8080 微处理器，时钟频率为 800kHz~3MHz。在 8 位微处理器中，最有影响的有四种产品：Intel8080 系列，Motorala 公司的 6800 系列，Zilog 公司的 Z80 及

Rockwell 公司的 6502。它们广泛应用于事务管理、工业控制、教育、通信等行业。

1978 年 Intel 公司推出 16 位的 8086, 后来又推出准 16 位的 8088, 成为个人计算机的主流 CPU。16 位微处理器中最有代表性的是 Intel8086/8088 和 80186、80286, Motorola 公司的 M68000 等, 时钟频率 4.77~16MHz。

1985 年, Motorola 公司首先推出 32 位微处理器 68020, Intel 公司于同年推出 80386 与之竞争。1989 年 Motorola 公司又宣布一种新的 32 位处理器 68040, 几天之后 Intel 公司又生产出 80486, 其速度比 80386 快 3 倍, 时钟频率 16~66MHz。正是由于有了这些微处理器芯片, 再加上适当的系统配置, 才有了我们现在所说的 286、386、486 等微机系统。

1993 年以后, 典型产品是 Intel 公司的奔腾系列芯片和 AMD 的 K6 系列微处理器芯片。随着 Intel 公司的 MMX (Multi Media eXtended) 微处理器的出现, 使微机的发展在网络化、多媒体化和智能化等方面跨上了更高的台阶。

2000 年 3 月, AMD 与 Intel 分别推出了时钟频率达 1GHz 的 Athlon 和 Pentium III。

2000 年 11 月, Intel 又推出了 Pentium IV 微处理器, 主频 1.5GHz, 400MHz 的前端总线, 使用全新 SSE 2 指令集。

2002 年 11 月, Intel 推出的 Pentium IV 微处理器的时钟频率达到 3.06GHz, 而且微处理器还在不断地发展, 性能也在不断提升。

2006 年以后 Intel 推出了酷睿 2 (双核)、酷睿 2 (四核) 处理器。

Intel 公司在开始推出微处理器时, 最开始的奔腾 I~IV, 都是单核的, 后来又推出了双核、三核、四核, 甚至六核。因此英特尔后来的发展方向是生产多核处理器。

什么是多核处理器呢? 多核处理器即指基于单个半导体的一个处理器上拥有多个功能相同的处理器核心。换句话说, 将多个物理处理器核心整合入一个核中。

关于多核处理器, 从全球范围内看, 随着操作系统及应用软件对多核处理器的进一步支持及优化, AMD 及 Intel 公司为代表的处理器多核技术的推出, 将推动处理器多核化技术进一步发展。随着应用需求的扩大和技术的不断进步, 多核必将展示出其强大的性能优势。多核处理器是处理器发展的必然趋势, 无论是移动与嵌入式应用、桌面应用还是服务器应用, 都将采用多核的架构, 多核技术应用前景广阔。

多核处理器标志着计算技术的一次重大飞跃。这一重要进步是计算机应用者面对飞速增长的数字资料和互联网的全球化趋势, 开始要求处理器提供更多便利和优势。多核处理器, 较之当前的单核处理器, 能带来更多的性能和工作效率, 因而最终将成为一种广泛普及的计算模式。多核处理器还将在推动 PC 安全性和虚拟技术方面起到关键作用, 虚拟技术的发展能够提供更好的保护、更高的资源使用率和更可观的商业市场价值。计算机应用者也将比以往拥有更多的途径获得更高性能, 从而提高家用 PC 和数字媒体计算系统的使用。

目前, Intel 公司推出的酷睿 2E8600 双核 CPU, 主频 3.33GHz; 酷睿 i7-965Extreme 四核 CPU、主频 3.2GHz。AMD 公司推出的 AMD 双核速龙 64 X2 6000 + AM2(盒), 主频 3.1GHz; AMD Phenom9600 四核 CPU, 主频 2.3GHz。

## 1.2 微型计算机的分类

微型计算机的分类方法有多种。按微处理器的位数，可分为 1 位机、4 位机、8 位机、16 位机、32 位机和 64 位机等。按结构外形，分为单片机、单板计算机、台式微机和笔记本式微机。

单片机，它仅由一块超大规模集成电路组成，CPU、存储器、I/O 接口电路和总线制作在一块很小的芯片上。使用简单的开发装置可以对它进行在线开发。单片机在智能化仪器仪表、家用电器及其他各种嵌入式系统中获得了广泛的应用。

单板计算机的 CPU 是一块单独的大规模集成电路芯片，存储器和 I/O 接口电路也都是片或几片大规模集成电路芯片。这些芯片加上若干附加逻辑电路和简单的键盘/数码显示器，并装在一块印刷电路板上，便构成一个单板计算机。单板计算机结构简单，价格低廉，性能较好，常用做过程控制和各种仪器、仪表装置的控制部件。

台式机，系指由 CPU 芯片、存储器芯片、I/O 接口电路、I/O 适配器和必要的外部设备（键盘、CRT 显示器、磁盘/光盘驱动器等）组成的整机系统。CPU、ROM、RAM、I/O 接口都装在系统板（又叫主板）上。系统板上另有一些扩展插槽，用于插入存储板和 I/O 适配板，以扩充存储器容量和增加外设。系统板、扩充板、磁盘/光盘驱动器和系统电源等一起装在一个机箱中，称之为主机；外加一个键盘、一个 CRT 显示器，便构成了一台完整的微机。这种微机既可作为通用机，用于科学计算和数据处理；也可作为专用机，用于实时控制和管理等。

笔记本式微机是一种体积小、重量极轻，但功能又很强的便携式完整微机，通常装放在一个公文包式的小盒中。从笔记本式微机又衍生出掌上微机和膝上微机。

## 1.3 计算机的应用领域

随着科学技术的发展，计算机应用越来越广泛，以致很难逐一介绍。计算机应用大体可概括为科学计算、数据处理、实时控制、CAD 和智能模拟、通信和文字处理、信息网络化等几大类。

### 1. 科学计算

科学计算是计算机应用的一个十分重要的领域。用于快速解决科学技术和工程设计中存在的大量的数学计算问题。例如，卫星发射中，卫星轨道的计算、发射参数的计算、空气动力学计算等，都需要高速计算机完成。

### 2. 数据处理

数据处理已成为计算机应用的一个重要领域。例如，利用数据库系统软件，实现工资管理系统、人事档案管理系统、工厂管理系统等；利用计算机网络技术联网，实现信息资源共享，提高工作效率和工作质量。

### 3. 实时控制

实时控制是计算机在过程控制方面的重要应用。实时，系指计算机的运算与控制时间与被控制过程的真实时间相适应。通过计算机对工业生产的实时控制，实现工业生产

全自动化。

#### 4. 计算机辅助设计

为了提高设计质量，缩短设计周期，提高设计自动化水平，人们借助于计算机进行设计，称为计算机辅助设计（CAD, Computer Aided Design）。目前，在船舶设计、飞机设计、汽车设计和建筑工程设计等行业中，均已使用计算机辅助设计系统。

#### 5. 通信和文字处理

计算机在通信和文字处理方面的应用，越来越显示出巨大的潜力。依靠计算机网络存储和传送信息，将多台计算机、通信工作站和终端组成网络，实现信息交换、信息共享、前端处理、文字处理、语言和影像输入/输出等，是实现办公自动化、电子邮政、计算机出版等新技术的必由之路。

#### 6. 信息网络化

目前我国，个人计算机已经开始进入家庭，这标志着我国计算机普及将进入一个新阶段。利用卫星通信网和光导纤维网实现计算机网络化和信息双向交流，应用多媒体技术普及计算机的使用。

## 1.4 计算机硬件基础

### 1.4.1 计算机中数的表示和运算

计算机中使用的数据一般可以分为两大类：数值数据与字符数据。数值数据常用于表示数的大小与正负；字符数据则用于表示非数值的信息，如：英文、汉字、图形、语音等数据。数据在计算机中是以器件的物理状态（开、关状态）来表示的，因此，各种数据在计算机中都是用二进制编码的形式来表示的。

#### 1. 进位计数制

按进位的原则进行计数的方法，称为进位计数制。

例如，在十进位计数制中，是根据“逢十进一”的原则进行计数的。

一个十进制数，它的数值是由数码 0、1、…、9 来表示的。数码所处的位置不同，代表数的大小也不同。从右面起的第一位是个位，第二位是十位，第三位是百位，第四位是千位……“个、十、百、千……”在数学上叫做“位权”或“权”。每一位上的数码与该位“位权”的乘积表示该位数值的大小。另外，十进位计数制中的 10，称为基数。基数为 10 的进位计数制按“逢十进一”的原则进行计数。

“位权”和“基数”是进位计数制中的两个要素。

在微机中，常用的是十进制、二进制、十六进制，它们对应的关系见表 1-1。

表 1-1 十进制数、二进制数、十六进制数的关系

十进制数	二进制数	十六进制数
00	0000	0
01	0001	1
02	0010	2

续表

十进制数	二进制数	十六进制数
03	0011	3
04	0100	4
05	0101	5
06	0110	6
07	0111	7
08	1000	8
09	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

### （1）十进制数

十进制数 563.62 可表示为：

$$(563.62)_{10} = 5 \times (10)^2 + 6 \times (10)^1 + 3 \times (10)^0 + 6 \times (10)^{-1} + 2 \times (10)^{-2}$$

一般来说，任意一个十进制数  $N$  可表示为：

$$\begin{aligned} (N)_{10} &= K_{n-1} \times 10^{n-1} + K_{n-2} \times 10^{n-2} + \cdots + K_1 \times 10^1 + K_0 \times 10^0 + K_{-1} \times 10^{-1} + K_{-2} \times 10^{-2} + \cdots + K_{-m} \times 10^{-m} \\ &= \sum_{i=-m}^{n-1} K_i \times 10^i \end{aligned}$$

式中， $m$ 、 $n$  均为正整数， $m$  表示小数部分的位数， $n$  表示整数部分的位数； $K_i$  是 0~9 中的某一个， $10^i$  是权。

### （2）二进制数

二进制数的基数是 2，即“逢二进一”，它使用数字 0 和 1 两个数码。利用 0 和 1 可以表示开关的通、断状态。其表示方法如下：

$$(10111.101)_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$

也可以将一个二进制数表示为：

$$\begin{aligned} (N)_2 &= K_{n-1} \times 2^{n-1} + K_{n-2} \times 2^{n-2} + \cdots + K_1 \times 2^1 + K_0 \times 2^0 + K_{-1} \times 2^{-1} + K_{-2} \times 2^{-2} + \cdots + K_{-m} \times 2^{-m} \\ &= \sum_{i=-m}^{n-1} K_i \times 2^i \end{aligned}$$

式中， $n$  表示整数部分的位数， $m$  表示小数部分的位数； $K_i$  是 1 或 0， $2^i$  是权。

### （3）十六进制数

十六进制数是 0~9、A~F，其中 A~F 分别代表 10~15；其基数为 16，即“逢十六进一”。其表示方法如下：

$$(2AC7.1F)_{16} = 2 \times 16^3 + 10 \times 16^2 + 12 \times 16^1 + 7 \times 16^0 + 1 \times 16^{-1} + 15 \times 16^{-2}$$

## 2. 不同进制数之间的转换

### (1) 十进制数与二进制数之间的转换

#### ① 十进制整数转换成二进制整数

十进制整数转换成二进制整数，通常采用“除2取余法”。所谓除2取余法，就是将已知十进制数反复除以2，若每次相除之后余数为1，则对应于二进制数的相应位为1；余数为0，则相应位为0。第一次相除得到的余数是二进制数的低位，最后一次相除的余数是二进制数的高位。从低位到高位逐次进行，直到商为0。最后一次相除所得的余数为 $K_{n-1}$ ，则 $K_{n-1}K_{n-2}\cdots K_1K_0$ 即为所求之二进制数。

【例 1-1】 将 $(215)_{10}$ 转换成二进制整数。其全过程可表示如下：

2	215	
2	107	..... 余数为1
2	53	..... 余数为1
2	26	..... 余数为1
2	13	..... 余数为0
2	6	..... 余数为1
2	3	..... 余数为0
2	1	..... 余数为1
	0	..... 余数为1

所以  $(215)_{10} = (K_7K_6K_5K_4K_3K_2K_1K_0)_2 = (11010111)_2$

#### ② 十进制纯小数转换成二进制纯小数

十进制纯小数转换成二进制纯小数，通常采用“乘2取整法”。所谓乘2取整法，就是将已知十进制纯小数反复乘以2，每次乘以2之后，所得新的整数部分为1，相应位为1；如果整数部分为0，则相应位为0。从高位向低位逐次进行，直到满足精度要求或乘以2后的小数部分为0为止。最后一次乘以2所得的整数部分为 $K_{-m}$ 。转换后，所得的纯二进制小数为 $0.K_{-1}K_{-2}\cdots K_{-m}$ 。

【例 1-2】 将 $(0.6531)_{10}$ 转换成纯二进制小数。转换过程如下：

0.6531	
×) 2	
0.3062	..... 整数部分为1 $K_{-1}$
×) 2	
0.6124	..... 整数部分为0 $K_{-2}$
×) 2	
0.2248	..... 整数部分为1 $K_{-3}$
×) 2	
0.4496	..... 整数部分为0 $K_{-4}$
×) 2	
0.8992	..... 整数部分为0 $K_{-5}$
×) 2	
0.7984	..... 整数部分为1 $K_{-6}$



如只取六位小数能满足精度要求，则得：

$$(0.6531)_{10} = (0.K_{-1}K_{-2}\cdots K_{-m})_2 \approx (0.K_{-1}K_{-2}K_{-3}K_{-4}K_{-5}K_{-6})_2 = (0.101001)_2$$

可见，十进制纯小数不一定能转换成完全等值的二进制纯小数。当遇到这种情况时，可根据精度要求，取近似值。

### ③ 十进制数转换成二进制数

【例 1-3】 将 $(215.6531)_{10}$ 转换为二进制数。

$$\text{因为 } (215)_{10} = (11010111)_2 \quad (0.6531)_{10} \approx (0.101001)_2$$

$$\text{所以 } (215.6531)_{10} \approx (11010111.101001)_2$$

### ④ 二进制数转换成十进制数

【例 1-4】 将二进制数 11001.1001 转换成十进制数。

$$\begin{aligned} (11001.1001)_2 &= 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} \\ &= 16 + 8 + 1 + 0.5 + 0.0625 \\ &= (25.5625)_{10} \end{aligned}$$

## (2) 二进制数与十六进制数之间的转换

### ① 二进制数转换成十六进制数

对于二进制整数，只要自右向左将每四位二进制数分为一组，不足四位时，在左边添 0，补足四位；对于二进制小数，只要自左向右将每四位二进制数分为一组，不足四位时，在右边添 0，补足四位。然后将每组用相应的十六进制数代替，即可完成转换。

【例 1-5】 把 $(101101101.0100101)_2$ 转换成十六进制数。

$$\begin{array}{cccccc} (0001 & 0110 & 1101 & . & 0100 & 1010)_2 \\ \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow \\ (1 & 6 & D & . & 4 & A)_{16} \end{array}$$

$$\text{即 } (101101101.0100101)_2 = (16D.4A)_{16}$$

### ② 十六进制数转换成二进制数

将十六进制数转换成二进制数，只要将每一位十六进制数用四位相应的二进制数表示，即可完成转换。

【例 1-6】 将 $(1863.5B)_{16}$ 转换成二进制数。

$$\begin{array}{cccccc} (1 & 8 & 6 & 3 & . & 5 & B)_{16} \\ \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow \\ (0001 & 1000 & 0110 & 0011 & . & 0101 & 1011)_2 \end{array}$$

$$\text{即 } (1863.5B)_{16} = (1100001100011.01011011)_2$$

## 3. 带符号数的表示及运算

在应用中数字有正有负，计算机中该如何表示呢？计算机中所能表示的数或其他信息都是数字化的，即用数字 0 或 1 来表示数的正负号。一个数的最高位为符号位，若该位为 0，则表示正数；若该位为 1，则表示负数。

【例 1-7】 用八位二进制数表示+20 和-20 分别为：

$$\begin{array}{ll} +20 & 00010100 \\ -20 & 10010100 \end{array}$$



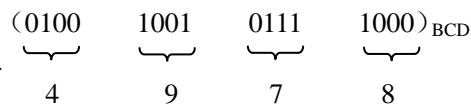


表 1-2 列出一部分编码关系。

表 1-2 BCD 编码表

十进制数	8421BCD 码	十进制数	8421BCD 码
0	0000	8	1000
1	0001	9	1001
2	0010	10	0001 0000
3	0011	11	0001 0001
4	0100	12	0001 0010
5	0101	13	0001 0011
6	0110	14	0001 0100
7	0111	15	0001 0101

(2) 字符、文字的编码

由于计算机内部存储、传送及处理的信息只有二进制信息，因此各种文字、符号也就必须用二进制数编码表示。

ASCII 码是美国标准信息码，它采用七位二进制数 ( $b_7b_6b_5b_4b_3b_2b_1$ ) 表示 128 个符号。例如：0~9 的 ASCII 码为 30H~39H；大写字母 A~Z 的 ASCII 码为 41H~5AH。ASCII 编码见表 1-3。

表 1-3 ASCII 编码

列	0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---	---

行	MSD LSD	000	001	010	011	100	101	110	111
0	0000	NUL	DEL	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	\$	4	D	T	d	t
5	0101	ENG	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	↑	n	~
F	1111	SI	VS	/	?	O	←	o	DEL

## 5. 位、字节和字的基本概念

计算机存储信息的最小单位是“位”，是指二进制数中的一个数位，一般我们称之为比特（bit），其值为“0”或“1”。

在计算机中，我们常使用字节（Byte）作为计量单位，一个字节由8个二进制位组成，其最小值为0，最大值为 $(11111111)_2 = (FF)_{16} = 255$ 。一个字节对应计算机的一个存储单元，它可存储一定的内容，例如存储一个英文字母“A”的编码，其对应的内容为“01000001”。由于计算机中有很多个存储单元，我们还需要将它们的地址进行编码。若存储器容量为1024字节（简称为1K字节），就需要用 $2^{10} = 1024$ 个地址编码。有时我们又将两个字节合称为一个字（Word）。

## 6. 算术与逻辑运算

### （1）算术运算

算术的基本运算有四种算法：加、减、乘、除。

#### ① 二进制加法

二进制加法的运算规则为：

$$0+0=0 \quad 0+1=1+0=1 \quad 1+1=0 \text{（进位1）} \quad 1+1+1=1 \text{（进位1）}$$

例如：

$$\begin{array}{r} 1101 \\ +) \quad 1011 \\ \hline 11000 \end{array}$$

#### ② 二进制减法

二进制减法的运算规则为：

$$0-0=0 \quad 1-1=0 \quad 1-0=1 \quad 0-1=1 \text{（有借位）}$$

例如：

$$\begin{array}{r} 1101 \\ -) \quad 0111 \\ \hline 0110 \end{array}$$

#### ③ 二进制乘法

二进制乘法的运算规则为：

$$0 \times 0 = 0 \quad 0 \times 1 = 0 \quad 1 \times 0 = 0 \quad 1 \times 1 = 1$$

例如：

$$\begin{array}{r} 1101 \\ \times \quad 110 \\ \hline 0000 \\ 1101 \\ 1101 \\ \hline 1001110 \end{array}$$

#### ④ 二进制除法

例如：

$$\begin{array}{r}
 \phantom{101}101 \quad \text{商} \\
 101 \overline{) 11011} \\
 \underline{101} \phantom{00} \\
 111 \phantom{0} \\
 \underline{101} \phantom{0} \\
 10 \phantom{0} \quad \text{余数}
 \end{array}$$

即

$$11011 \div 101 = 101 \text{ 余 } 10$$

## （2）逻辑运算

布尔代数主要研究如何对事物内部的逻辑关系进行表达和运算。逻辑数据只有两个值，“真”和“假”。逻辑运算主要有以下几种。

### ① 与运算（ $Y=A \wedge B$ ）

与运算也称为逻辑乘法运算，通常用符号“ $\cdot$ ”或“ $\wedge$ ”、“ $\times$ ”表示。它的运算规则为：

$$0 \wedge 0 = 0 \quad 1 \wedge 0 = 0 \quad 0 \wedge 1 = 0 \quad 1 \wedge 1 = 1$$

这种运算的结果也可以归纳为两句话：二者皆为真则结果为真，有一为伪则结果必伪。

例如：设  $A=11001010$ ， $B=00001111$ ，则

$$\begin{array}{r}
 11001010 \\
 \wedge) 00001111 \\
 \hline
 00001010
 \end{array}$$

即

$$Y=A \wedge B=00001010$$

由此可见，用“0”去和一个数位相“与”，其运算结果为 0；用“1”去和一个数位相“与”，就是将此数位“保存”下来。

### ② 或运算（ $Y=A \vee B$ ）

“或”运算也称逻辑加法，常用符号“ $+$ ”或“ $\vee$ ”表示。它的运算规则为：

$$0 \vee 0 = 0 \quad 0 \vee 1 = 1 \quad 1 \vee 0 = 1 \quad 1 \vee 1 = 1$$

上述四个式子与一般加法不同，Y 的值只能有两个数值 0 或 1。上面四个式子可以归纳成两句话：两者皆伪则结果必伪，有一为真则结果为真。

例如：设  $A=10101$ ， $B=11011$ ，则：

$$\begin{array}{r}
 10101 \\
 \vee) 11011 \\
 \hline
 11111
 \end{array}$$

即

$$Y=A \vee B=11111$$

注意：1“或”1 等于 1，是没有进位的。

### ③ 反运算（ $Y=\bar{A}$ ）

反运算又称非运算，为逻辑否定。如果一事物的性质为 A，则经过“反”运算之后，其性质必与 A 相反，用表达式表示为： $Y=\bar{A}$ 。其运算规则为：

$\bar{0}=1$  读成“非0等于1”

$\bar{1}=0$  读成“非1等于0”

当A为多位数时,如  $A=A_1A_2A_3\cdots A_n$ , 则其“逻辑反”为:  $Y=\bar{A}_1\bar{A}_2\bar{A}_3\cdots\bar{A}_n$ 。

例如: 设  $A=1101000$ , 则:  $Y=\bar{A}=0010111$ 。

#### ④ 异或运算 ( $Y=A\oplus B$ )

异或运算通常用符号“ $\oplus$ ”表示。它的运算规则为:

$$0\oplus 0=0 \quad 0\oplus 1=1 \quad 1\oplus 0=1 \quad 1\oplus 1=0$$

同样, 异或运算结果也可归纳成两句话: 只要两逻辑变量相同, 则异或运算的结果就为0; 当两个逻辑变量不同时, 异或运算的结果为1。

当A或B为二进制数时, 则进行异或运算时, 各对应位分别进行异或运算。

例如: 设  $A=1010$ ,  $B=1101$ , 则:

$$\begin{array}{r} 1010 \\ \oplus \quad ) \quad 1101 \\ \hline 0111 \end{array}$$

即  $Y=A\oplus B=0111$

### 1.4.2 微型计算机的基本组成电路

计算机是由若干个基本电路单元组成的。本小节对微机中最常见的基本电路做一简单介绍, 这些电路是组成计算机的硬件基础。

#### 1. 常用逻辑电路

逻辑电路是计算机执行运算、控制功能所必需的电路, 是计算机的基本单元电路。

逻辑电路中, 其输入和输出只有两种状态, 即高电平和低电平。通常以逻辑1和0表示电平的高低。下面简单介绍常用逻辑电路的逻辑符号和逻辑功能。具体电路可参看有关书籍。

##### (1) 与门

与门是一个能够实现逻辑乘运算的、具有多端输入/单端输出的逻辑电路。图1-1(a)所示是一个二输入的与门, 其逻辑函数式是:

$$Y=A\wedge B, \text{ 或 } Y=A\cdot B$$

常用逻辑电路的真值表如表1-4所示。

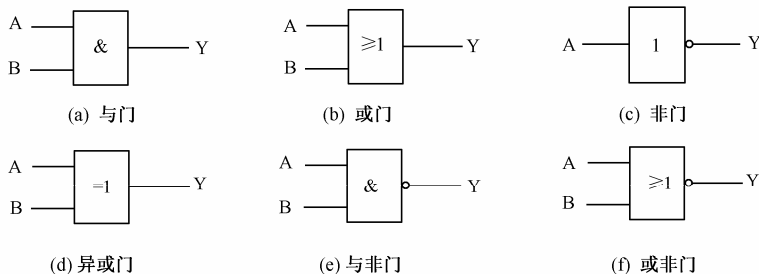


图1-1 常用逻辑单元图形符号

表 1-4 常用逻辑电路真值表

输 入		输 出				
A	B	与门	或门	非门	异或门	与非门
0	0	0	0	1	0	1
0	1	0	1	1	1	0
1	0	0	1	0	1	0
1	1	1	1	0	0	0

### (2) 或门

或门是一个能够实现逻辑加运算的、具有多端输入/单端输出的逻辑电路。图 1-1 (b) 就是一个二输入的或门，其逻辑函数式是：

$$Y=A \vee B$$

### (3) 非门

非门是一个能够完成逻辑非运算的、具有单端输入/单端输出的逻辑电路。图 1-1 (c) 就是非门电路，其逻辑函数式是：

$$Y=\overline{A}$$

### (4) 异或门

异或门是一个能够完成逻辑异或运算的多端输入/单端输出的逻辑电路。图 1-1 (d) 就是一个二输入的异或门，其逻辑函数式是：

$$Y=A \oplus B$$

### (5) 与非门

与非门是一个能够完成逻辑与非运算的多端输入/单端输出的逻辑电路。图 1-1 (e) 就是一个二输入的与非门，其逻辑函数式是：

$$Y=\overline{A \wedge B}$$

### (6) 或非门

或非门是一个能够完成逻辑或非运算的多端输入/单端输出的逻辑电路。图 1-1 (f) 就是一个二输入的或非门，其逻辑函数式是：

$$Y=\overline{A \vee B}$$

## 2. 触发器

触发器是计算机记忆装置的基本单元，它具有把以前的输入“记忆”下来的功能，一个触发器能储存一位二进制代码。下面简要介绍几种计算机中常用的触发器。

### (1) RS触发器

RS 触发器的逻辑符号如图 1-2 所示。它有两个输入端，两个输出端。其中 S 为置位信号输入端，R 为复位信号输入端；Q 和  $\overline{Q}$  为输出端。规定 Q 为高电平， $\overline{Q}$  为低电平时，该触发器为 1 状态；反之为 0 状态。其真值表如表 1-5 所示。

### (2) D触发器

D 触发器又称数据触发器，它的逻辑符号如图 1-3 所示。R、S 分别为置 0、置 1 端，触发器的状态由时钟脉冲到来时，D 端的状态决定。当 D=1 时，触发器为 1 状态；反之为

0 状态。其真值表如表 1-6 所示。

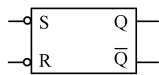


图 1-2 RS 触发器的逻辑符号

表 1-5 RS 触发器真值表

输入		输出	
S	R	Q	$\bar{Q}$
0	0	不确定	
0	1	1	0
1	0	0	1
1	1	保持不变	

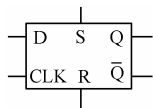


图 1-3 D 触发器的逻辑符号

表 1-6 D 触发器真值表

输 入	输 出
D	Q
0	0
1	1

(3) JK 触发器

JK 触发器的逻辑符号如图 1-4 所示。R、S 分别为直接置 0 端和置 1 端。K 为同步置 0 输入端，J 为同步置 1 输入端。其真值表如表 1-7 所示。

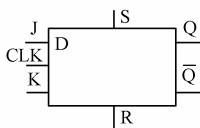


图 1-4 JK 触发器的逻辑符号

表 1-7 JK 触发器真值表

输 入		输 出
J	K	Q
0	0	不变
0	1	0
1	0	1
1	1	翻转

JK 触发器的逻辑功能比较全面，因此在各种寄存器、计数器、逻辑控制等方面的应用最为广泛。但在某些情况下，如二进制计数、移位、累加等，多用 D 触发器。因为 D 触发器的线路简单，所以大量应用于移位寄存器等方面。

3. 寄存器

寄存器是由触发器组成的。一个触发器就是一个一位寄存器。多个触发器就可以组成一个多位寄存器。

(1) 锁存器

它是用以暂存某个数据，以便在适当的时间节拍将数据输入或输出到其他记忆元件中去。图 1-5 是一个并行输入与并行输出的四位锁存寄存器的电路原理图，它由四个 D 触发器组成。

开始时，先在清 0 端加清 0 脉冲，把各触发器置 0，即 Q 端为 0。然后将数据加到触发器的 D 输入端，在 CLK 时钟信号作用下，输入端的信息就保存在各触发器中 ( $D_0 \sim D_3$ )。



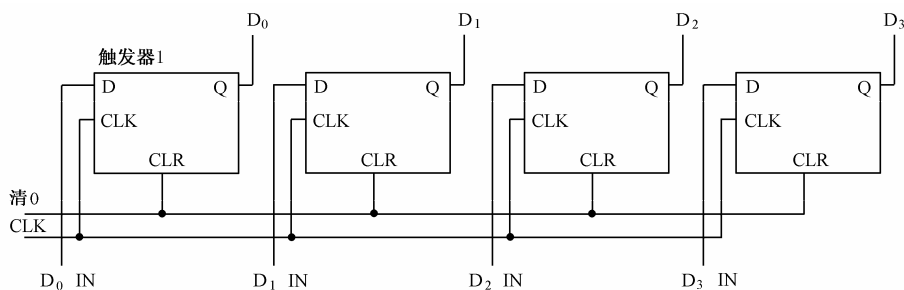


图 1-5 四位锁存寄存器的电路原理图

### （2）移位寄存器(Shifting Register)

移位寄存器能将所储存的数据逐位向左或向右移动，以达到计算机运行过程中所需的功能。图 1-6 所示即为一串行输入四位移位寄存器电路。

开始时，先在清 0 端加清 0 脉冲，使触发器输出置 0。将第一个数据  $D_0$  加到触发器 1 的串行输入端，在第一个 CLK 脉冲到达时  $Q_0 = D_0$ ,  $Q_1 = Q_2 = Q_3 = 0$ ；然后将第二个数据  $D_1$  加到串行输入端，在第二个 CLK 脉冲到达时， $Q_0 = D_1$ ,  $Q_1 = D_0$ ,  $Q_2 = Q_3 = 0$ 。依此类推，当第四个 CLK 来到之后，各输出端分别是， $Q_0 = D_3$ ,  $Q_1 = D_2$ ,  $Q_2 = D_1$ ,  $Q_3 = D_0$ 。输出数据可用串行的形式取出，也可用并行的形式取出。

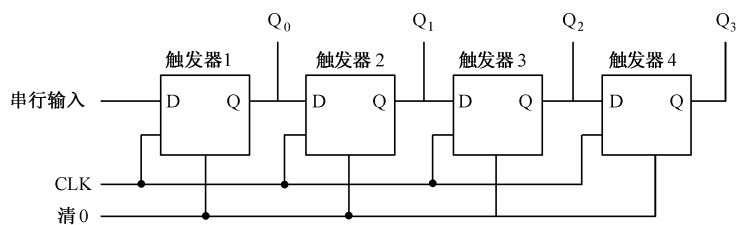


图 1-6 四位移位寄存器

### （3）计数器（Counter）

计数器也是由若干个触发器组成的寄存器，它的特点是能够对时钟脉冲进行计数等。

图 1-7 所示为计数器的电路原理图。图中各位的 J、K 输入端都是悬浮的。这相当于 J、K 输入端都是置 1 的状态，即各位都处于准备翻转的状态。只要时钟脉冲边沿一到，最右边的触发器就会翻转，即由 0 转为 1 或由 1 转为 0。

图 1-7 中的计数器是四位的，因此可以计 0~15 的数。如果要计更多的数，就需要加位数，如八位计数器可计 0~255 的数，16 位则可以计 0~65535 的数。

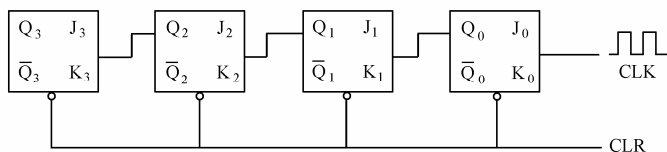


图 1-7 计数器的电路原理图

### （4）三态门（三态寄存器）

微型计算机的总线“共享”总线通道，能正确实现信息源与信息目的地的对应传输。

采用三态门电路（或称三态门）把部件与总线相连。当部件不工作时，与总线相连的三态输出电路处于高阻态，犹如与总线断开一样，对总线不产生影响。

所谓三态是指输出电路具有 0 态（开通，传输“0”）、1 态（开通，传输“1”）、高阻态（断开/悬浮输出）。图 1-8 给出了总线结构上广泛采用的单向和双向三态门的电路原理。三态门“开”或“关”的控制信号一般由微处理器发出。双向三态门由两个单向三态门构成，又称做双向电子开关。工作时，用两个单向三态门互斥的控制端信号来选通传输方向。

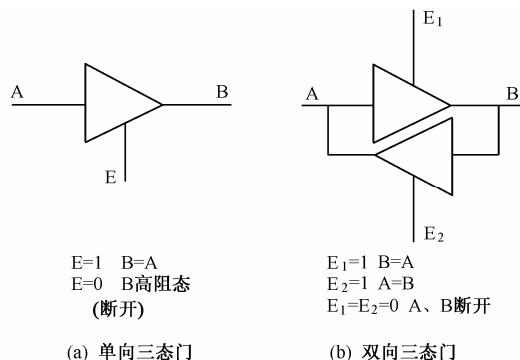


图 1-8 总线结构上采用的三态电路

三态门一般具有较高的输入阻抗和较低的输出阻抗，可以改善传输特性，故对传输数据起到缓冲作用，同时能对传输的数据进行功率放大，具有一定的驱动能力。所以三态门电路还被称为数据缓冲/驱动电路。

当控制端  $E=1$  时，输出等于输入，此时总线由该器件驱动，总线上的数据由输入数据决定。当  $E=0$  时，输出端呈高阻抗状态，该器件对总线不起作用。当寄存器的输出端接至三态门，再将三态门的输出端与总线连接起来，就构成三态输出的缓冲寄存器。图 1-9 所示即为一个四位的三态输出缓冲寄存器。因图中采用的是单向三态门，所以数据只能从寄存器输出至数据总线。如果要实现双向传送，则要用双向三态门，见图 1-8 (b)。

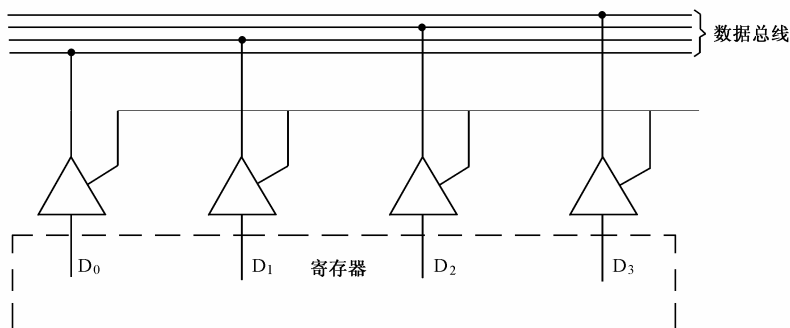


图 1-9 三态输出缓冲寄存器

### (5) 译码器

微机中广泛采用地址译码器来对存储器或输入/输出设备进行选择，操作其工作。例如，CPU 在给出存储单元的地址后，存储器要根据该地址选择对应的存储单元，这个过程

叫做地址译码。设存储单元的地址码为  $n$  位二进制数，存储单元的总数为  $N$  个，则有  $N=2^n$ 。地址译码就是根据  $n$  位地址码，在  $N$  个存储单元中选中对应的一个存储单元进行读/写的。这个选择工作是由地址译码电路来完成的。

译码电路的功能是，对输入的一个二进制数码经“翻译”后产生一个对应的输出有效信号。 $N$  位二进制数有  $2^n$  个不同的编码组合，所以，译码电路有  $n$  个输入端， $2^n$  个输出端。译码电路工作时，在某一时刻， $2^n$  个输出中只能有一个（和当前输入的代码相对应的）输出信号为有效，其余均为无效。若以输出低电平“0”为有效，则高电平“1”表示无效。

图 1-10 中给出了 3-8 译码器（74LS138）的逻辑电路。表 1-8 的真值表有三个输入端  $A_0 \sim A_2$ ，八个输出端  $\overline{Y}_0 \sim \overline{Y}_7$ （低电平有效），三个选通信号（或称允许信号） $\overline{G}_{2A}$ 、 $\overline{G}_{2B}$  和  $G$ 。只有当  $\overline{G}_{2A}$ 、 $\overline{G}_{2B}$  为低电平， $G$  为高电平，即  $\overline{G}_{2A} \wedge \overline{G}_{2B} \wedge G=1$  时，译码器才能根据输入  $A_0 \sim A_2$  的组合进行译码。

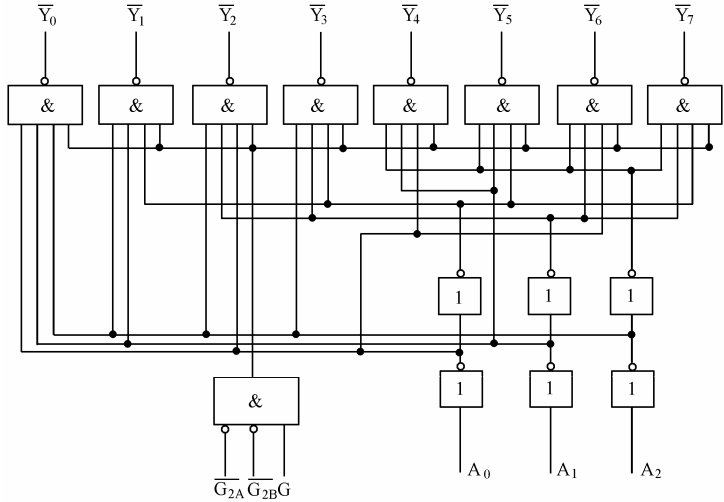


图 1-10 3-8 译码器 74LS138 的逻辑电路

表 1-8 3-8 译码器 74LS138 的真值表

输 入			输 出
译码控制		译码选择	
$\overline{G}$	$\overline{G}_{2B}$	$\overline{G}_{2A}$	
		C B A	
1	0	0	$\overline{Y}_0=0$ , 其余为 1
1	0	1	$\overline{Y}_1=0$ , 其余为 1
1	0	0	$\overline{Y}_2=0$ , 其余为 1
1	0	1	$\overline{Y}_3=0$ , 其余为 1
1	0	0	$\overline{Y}_4=0$ , 其余为 1
1	0	1	$\overline{Y}_5=0$ , 其余为 1
1	0	0	$\overline{Y}_6=0$ , 其余为 1
1	0	1	$\overline{Y}_7=0$ , 其余为 1
不是上述情况			全部输出为 1

## 1.5 微型计算机系统

### 1.5.1 微型计算机系统的组成

计算机硬件（Hardware）是指计算机系统中看得见、摸得着的物理实体，是组成一个计算机系统的物质基础。硬件系统的基本功能是，能够执行预先设计好的在相应指令系统中的各种指令。

计算机硬件系统的结构如图 1-11 所示，由存储器、控制器、运算器、输入设备和输出设备五大部分组成。控制器和运算器合在一起称为 CPU（中央处理器）。（内）存储器和中央处理器合在一起称为主机。在计算机硬件系统中不属于主机的设备都属于外部设备，或叫做外围设备，简称外设。主机和外设合在一起构成了计算机系统。通常，将一个仅由硬件组成的计算机称为“裸机”。

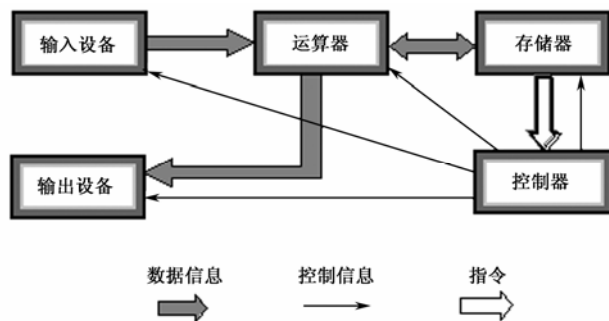


图 1-11 计算机硬件系统的结构

微型计算机的基本硬件结构也是由上述五部分组成的。

用大规模集成电路技术把运算器和控制器集成在一起，就构成了微型计算机的核心——微处理器，再配以大规模集成电路的主存储器芯片，通过接口电路将输入及输出设备连接起来，就组成了微型计算机的硬件系统，其结构如图 1-12 所示。

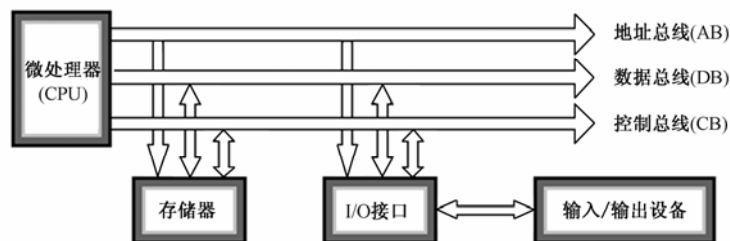


图 1-12 微型计算机结构图

一个完整的微型计算机系统由硬件系统和软件系统组成，如图 1-13 所示。微型计算机的基本硬件配置包括主机、键盘、磁盘驱动器、显示器等，软件配置包括操作系统、计算机语言、应用软件等。

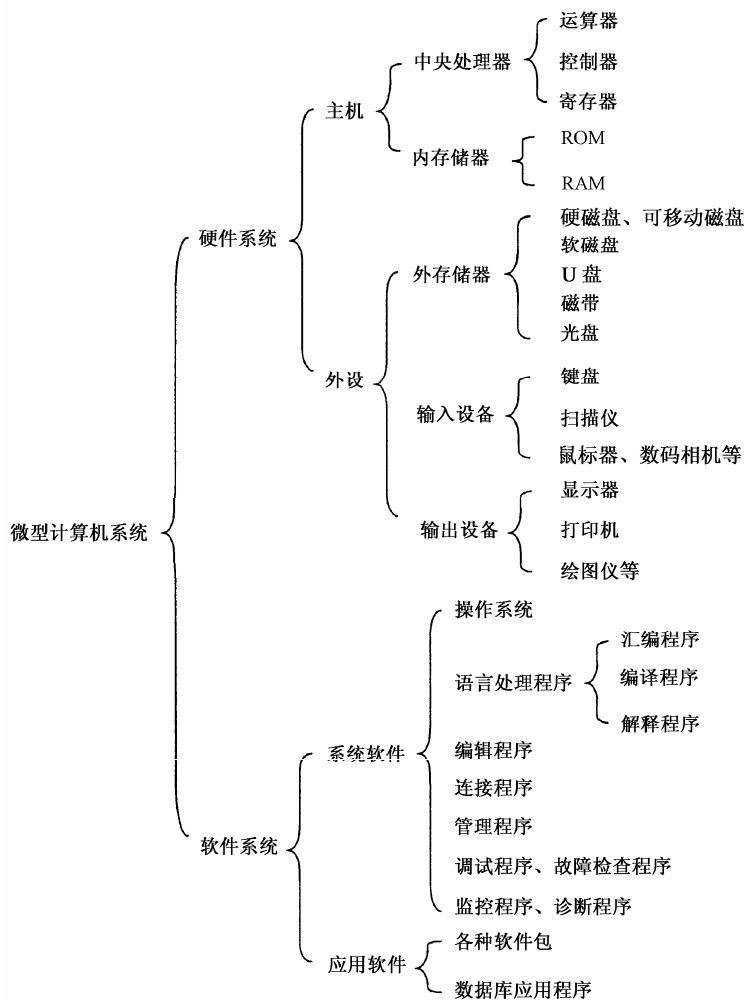


图 1-13 微型计算机系统结构示意图

微型计算机软件、硬件之间的关系是：

① 硬件是软件的物质基础。硬件是组成一个计算机的物质基础。任何软件都是建立在硬件基础之上的，离开硬件，软件不可能工作。

② 软件是硬件功能的扩充和完善。没有软件，硬件是废物一堆，有了软件，硬件才能正常运行并提高效率。软件是在硬件基础之上对硬件性能的扩充和完善。如果说硬件提供了使用工具，那么软件就为人们提供了使用的方法和手段，从而使人们不必了解机器本身就可以使用电子计算机，这就有利于计算机的推广和普及。

③ 软件和硬件的融合与转化。在微型计算机系统中，很多功能无法明确地说，哪些必须由软件来完成，哪些必须由硬件来完成。事实上，软件和硬件的界面是浮动的，某些计算机系统的功能既可以由硬件实现，也可以由软件实现，两者之间不存在一条固定不变的界限。某种系统功能如果用硬件实现，一般速度较快，但价格较高；如果用软件实现则价格较低，且较灵活，但速度可能会降低。总之，软件和硬件之间具有融合性和

转化性，在逻辑上是等价的。

### 1.5.2 微型计算机的基本结构

微型计算机的基本结构是由微处理器、存储器和 I/O 接口芯片，通过三条总线连接而成的。微型计算机结构上的最大的特点就是总线结构。所谓总线（BUS），是微型计算机中传送信息的一组通信线，它联系着多个信号源和多个接收部件，信号可以从多个信号源中的任意一个传递到接收部件中的任何一个。也就是说，构成微计算机的 CPU、存储器、I/O 接口都以平等的身份挂在总线上，它们按时间轮流使用总线，称为分时复用。所以，总线就像人体的神经一样牵动着全身，沟通着微型计算机的各个部分。

微型计算机的总线有三种：地址总线（Address Bus）、数据总线（Data Bus）和控制总线（Control Bus），分别为 AB、DB 和 CB。

#### 1. 地址总线AB

地址总线是用于传递存储单元或 I/O 端口地址信息的一组信号线。地址线由 CPU 发出（除 DMA 方式外），对存储单元和 I/O 端口（外部设备）进行寻址，所以它是单向并行传递的。8 位微处理器通常有 16 根地址线，记为  $A_0 \sim A_{15}$ ，能寻址的存储单元为： $2^{16}=65536=64K$ ；能对 I/O 端口寻址为： $2^8=256$  个。

#### 2. 数据总线DB

数据总线用于传送数据信息。对于 8 位微处理器，数据总线是 8 根，记为  $DB_0 \sim DB_7$ ，数据的传送可从 CPU 向存储器和 I/O 接口输出数据，有时又可由存储器和 I/O 接口向 CPU 输入数据。因此数据总线是双向并行传送信息的。

#### 3. 控制总线CB

控制总线用于传送各种控制命令，如定时脉冲、存储器和 I/O 接口的读/写控制、中断请求等。控制总线中的每一条控制线传送一种控制信号，因此是单向传送的。

各种标准的微处理器，原则上都有相同功能的数据总线和地址总线，它们的差别主要体现在控制总线上。正是由于控制总线具体特性的差异，使我们在使用接口芯片时，要考虑与微处理器是否兼容的问题。

## 1.6 微处理器的组成

微处理器的内部结构如图 1-14 所示。它主要由三个部分组成：算术逻辑单元 ALU（运算器）、指令部件（控制器）和寄存器组。微处理器中的各个部分之间采用总线连接，称之为内部总线。微处理器通过 AB（地址总线）、DB（数据总线）和 CB（控制总线）同其他外部部件连接，称之为外部总线。

微处理器在结构上有两个特点。一是采用内部总线。在有些微处理器中，两个操作数及运算结果用同一组内部总线分时传送，这是单总线结构（见图 1-14）。二是双总线结构（两个操作数用不同的内部总线分别传送）和三总线结构（两个操作数及运算结果分别用三组内部总线独立传送）。多总线结构运算速度快，布线复杂；单总线结构虽然速度要慢一些，但布线少，加工容易。目前，单总线结构比较流行。

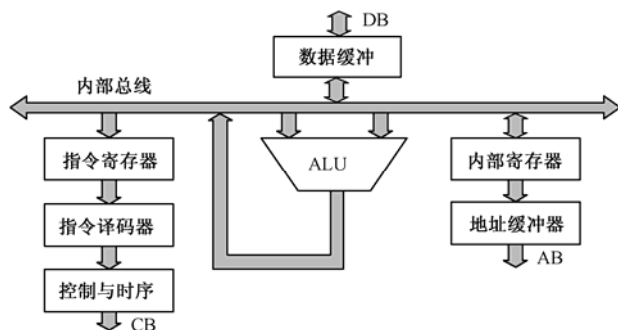


图 1-14 微处理器的内部结构

微处理器的第二个特点是采用多个内部寄存器。由于微型计算机采用外部总线和内部总线，使数据传送速度受到限制，因而，在操作中要尽量避免频繁地通过外部总线同存储器和输入/输出接口交换信息。在微处理器内部多设几个寄存器，用来保存操作数和中间结果，就可以大大提高运算速度，以弥补总线结构速度慢的不足。不过，寄存器的个数受到指令长度的限制。

## 1.7 微型计算机系统的主要性能指标

### 1. 字长

字长是指计算机的运算部件直接能处理的二进制数据的位数。字长越长，计算机的处理能力就越强，运算精度越高，指令功能越强，可寻址的存储空间也越大。所以，字长是评价计算机性能的一个非常重要的指标。微处理器的数据总线宽度一般与字长一致。微型计算机的字长一般为字节（8 位）的整数倍。早期的微型计算机的字长为 8 位，如 CROMEMCO；IBM PC/XT 字长为 16 位；386、486、奔腾及其兼容机的字长为 32 位；安腾的字长为 64 位。

### 2. 地址线

微处理器的地址处理能力与其地址线的数量有关，地址线数量决定可直接寻址的存储器空间范围，地址线多则寻址空间大。

### 3. 速度

速度是指计算机进行数值计算或信息处理的快慢程度。微型计算机的速度指标可用主频及运算速度评价。

主频又称时钟频率，是指微处理器工作时钟的频率，它在很大程度上决定了微处理器的运行速度，是决定微型计算机运行速度的重要指标之一。主频以兆赫兹为单位（MHz）。主频越高，微型计算机的运行速度越快。运算速度的单位为 MIPS（每秒百万指令数），这个指标较主频更能直观地反映微型计算机的运行速度。

一个运行速度快的系统，不仅要考虑处理器的时钟频率，还要考虑内存控制、磁盘驱动器及图形加速器的性能。

### 4. 指令系统

指令系统是指一台微处理器所能执行的全部指令，由于指令是规定微型计算机进行某

种操作的命令，因此指令系统在很大程度上决定了微处理器的工作能力。

### 5. 存储容量

微型计算机的处理能力不仅与字长和速度有关，而且在很大程度上还取决于存储系统的容量。存储系统主要包括主存和辅存（如磁盘、磁带）。存储容量以字节或字为单位。一个字节由 8 位二进制数组成。因为存储容量一般都很大，所以常以 KB（千字节）、MB（兆字节）或 GB（千兆字节）为单位， $1\text{KB}=1024\text{B}$ ， $1\text{MB}=1024\text{KB}=1048576\text{B}$ 。常见微型计算机主存容量有 128MB、256MB、512MB、1GB、2GB 等各档。

### 6. 兼容性

“兼容”是一个广泛的概念，这里主要指程序兼容。在前期微处理器上开发的程序在后期微处理器上照样可以运行，称之为向上兼容。兼容可使机器容易推广，对用户来说，又可减少软件开发的工作量。

## 1.8 微型计算机的一般工作过程

如前所述，微型计算机在硬件和软件相互配合之下才能工作。如果我们仔细注意微型计算机的工作过程就会发现，微型计算机为完成某种任务，总是将任务分解成一系列的基本动作，然后再一个一个地去完成每一个基本动作。当这一任务中所有的基本动作完成时，整个任务也就完成了。这是计算机工作的基本思路。

CPU 进行简单的算术运算或逻辑运算，或者从存储器取数，或者将数据存放于存储器，或者从接口取数，或者向接口送数，这些都是一些基本动作，也称为 CPU 的操作。

微处理器进行某种操作的代码叫做指令。前面已经提到，微处理器只认识由 0 电平和 1 电平组成的二进制编码，因此，指令就是一组由 0 和 1 构成的数字编码。微处理器在任何时刻只能进行一种操作。为了完成某种任务，需要把任务分解成若干个基本操作，明确完成任务的基本操作的先后顺序，然后用计算机可以认识的指令来编排完成任务的操作顺序。计算机的每一步操作都由特定的指令来指定，需按照事先编好的操作步骤，一步接一步地进行工作，从而达到预期的目的。这种完成某种任务的一组指令就称为程序，计算机的工作就是执行程序。

下面通过一个简单程序的执行过程，来对微型计算机的工作过程进行简要的介绍。随着本书的讲述，对计算机的工作原理将逐步得到深入理解。

**【例 1-9】**用微型计算机求解“ $7+10=?$ ”这样一个极为简单的问题必须利用指令告诉计算机该做的每一个步骤，以及先做什么，后做什么。其具体步骤就是：

第一步：采用助记符、操作数组成指令，根据题意编写程序。

```
MOV AL, 7
ADD AL, 10
HLT
```

程序中第一条指令将 7 放在 AL 中；第二条指令将 AL 中 7 加上 10，并将相加之和放在 AL 中；第三条指令是停机指令。当顺序执行完上述指令时，AL 中就存放着要求的结果。



第二步：将源程序翻译成机器语言（0或1），然后存入存储器，如下所示。

内存		
地址	10110000	} 第一条指令
	00000111	
	00000100	} 第二条指令
	00001010	
地址	11110100	第三条指令

第三步：微型计算机执行程序时，通过总线首先将第一条指令取进微处理器并执行它，然后取第二条指令，并执行第二条指令……计算机就是这样按照事先编排的顺序，依次执行指令。这里要再次强调，计算机只能识别机器代码，不认识助记符。因此，助记符编写的程序必须转换为机器代码才能被计算机直接识别。

## 习题

1. 按器件划分，各代电子计算机的特征是什么？
2. 简述微型计算机的发展趋势。
3. 简述微型计算机的基本组成。
4. 试说出微型计算机系统的主要性能指标。
5. 简要说明微处理器的一般组成及功能。
6. 说明微处理器的结构特点。
7. 试述微型计算机软件、硬件之间的关系。
8. 画图说明一个完整的微型计算机系统的组成。
9. 说明软件和硬件在逻辑上的等价性。
10. 微型计算机有哪三种总线，其作用是什么？
11. 将 $(255)_{10}$ 转换为二进制数。
12. 将 $(1111011111111011)_2$ 转换为十六进制数。
13. 逻辑运算包含几种运算？各是什么运算？
14. 已知  $A=1011$ ， $B=1000$ ，求  $Y=A \vee B$ 。
15. 三态门有什么特征？

# 第 2 章

## 80x86微处理器及其系统结构

### 教学目的和要求

本章主要介绍 8086/8088 的内部结构、外部结构和工作模式，以及 80x86 高档微处理器的体系结构。对于 80286、80386、80486 和 Pentium 微机的体系结构做一概括了解，重点掌握 8086/8088 的编程结构、8086/8088 的物理地址形成、两种工作模式。

## 2.1 8086/8088 的内部结构

8086 是 Intel 系列的 16 位微处理器，其芯片上集成有 2.9 万个晶体管，采用 HMOS 工艺制造，用单一的+5V 电源，时钟频率为 5~10MHz。

8086 有 16 根数据线和 20 根地址线，它既能处理 16 位数据，也能处理 8 位数据。可寻址的内存空间为 1MB。

Intel 公司在推出 8086 的同时，还推出了一种准 16 位微处理器 8088。8088 的内部寄存器、运算部件及内部数据总线都是按 16 位设计的，但外部数据总线只有 8 条。推出 8088 的主要目的是为了与当时已有的一套 Intel 外部设备接口芯片直接兼容使用。8086 与 8088 在寄存器结构、编程结构、存储器组织及 I/O 端口组织等方面几乎一样。

### 2.1.1 8086/8088 的编程结构

要掌握一个 CPU 的工作性能和使用方法，首先应该了解它的编程结构。这种结构与 CPU 内部的物理结构和实际布局是有区别的。8088 的编程结构图如图 2-1 所示。从图中可以看到，从功能上，可将 8086/8088 分为两部分，即总线接口部件 BIU（Bus Interface Unit）和执行部件 EU（Execution Unit）。

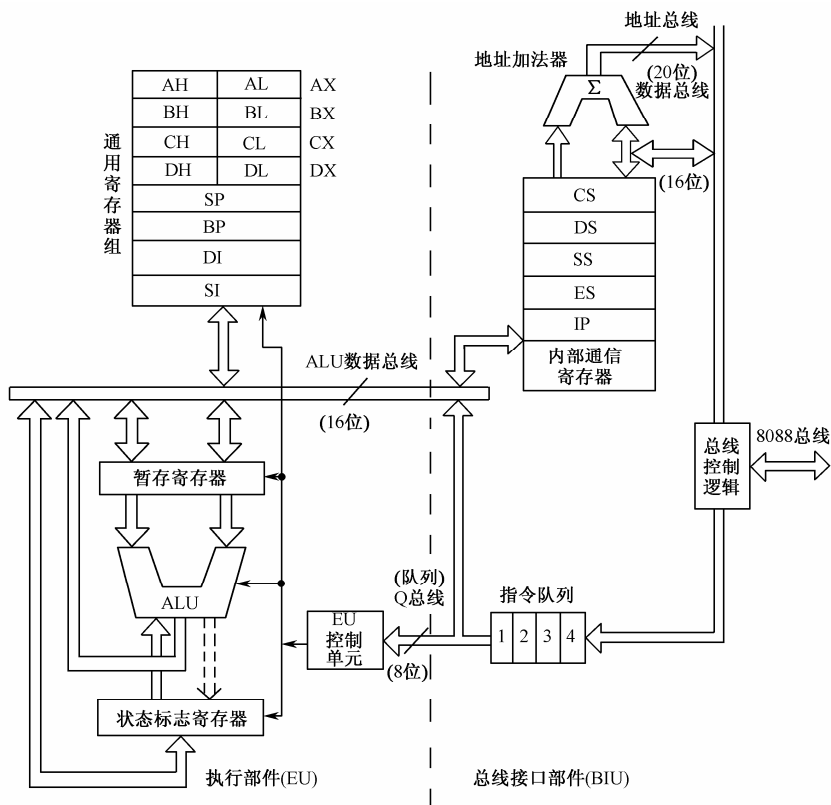


图 2-1 8088 CPU 的编程结构图

总线接口部件负责与存储器、外设端口传送数据。具体地讲，总线结构部件从内存取出指令送到指令队列；CPU 执行指令时，所需的操作数也由总线接口部件从指定的内存单元或外设端口取出，传送给执行部件去执行，反过来，执行部件的操作结果也通过总线接口部件传送到指定的内存单元或外设端口中。

总线接口部件由 4 部分组成：4 个段寄存器、指令指针寄存器 IP、20 位的地址加法器及 4 个字节的指令队列。

地址加法器的作用是产生 20 位地址。8086/8088 内部所有的寄存器都是 16 位的，8086/8088 可用 20 位地址去寻址 1MB 的内存空间，这就需要地址加法器根据 16 位寄存器提供的信息，计算出 20 位的物理地址。具体算法将在本节后面讲述存储器组织时，加以介绍。

对总线接口部件需要加以说明的一点是，8088 的指令队列为 4 个字节，而 8086 的指令队列为 6 个字节。不管是 8088，还是 8086，都会在执行指令的同时，从内存中取出下面一条指令或几条指令，取出的指令依次放在指令队列中，按顺序存放，并按顺序送到 EU 中去执行。

执行部件 EU 的功能是负责指令的执行。

执行部件包括：4 个数据寄存器、2 个指针寄存器、2 个变址寄存器、1 个状态标志寄存器和 1 个算术逻辑运算单元。

从编程结构可以看出，总线接口部件和执行部件是分开的。因此，每当 EU 执行一条指令，造成指令队列空出 2 个（对 8086）或空出 1 个（对 8088）指令字节时，BIU 会马上从内存中取出下面一条指令或几条指令，以填满它的指令队列。这样，一般情况下，CPU 在执行完一条指令后，便马上执行下一条指令，不像以往 8 位 CPU 那样，在执行完一条指令后，需要等待取出下一条指令，从而提高了 CPU 的效率。

### 2.1.2 8086/8088 的寄存器结构

8086/8088 的寄存器结构如图 2-2 所示。

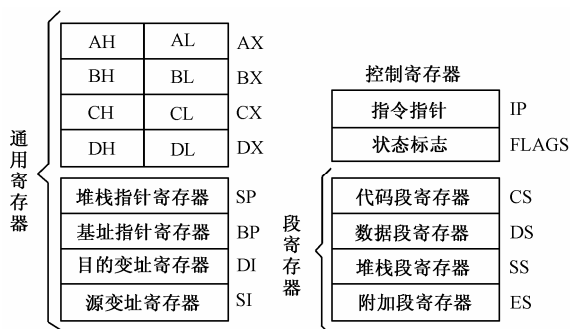


图 2-2 8088/8086 的寄存器结构

#### 1. 通用寄存器

8086/8088 的 CPU 有 8 个 16 位的通用寄存器，它们又被分为两组：

### （1）数据寄存器

4 个数据寄存器是：累加器 **AX**、基址寄存器 **BX**、计数寄存器 **CX** 及数据寄存器 **DX**。数据寄存器的特点是，这 4 个 16 位的寄存器可分为高 8 位（**AH**、**BH**、**CH** 和 **DH**）和低 8 位（**AL**、**BL**、**CL** 和 **DL**），这两组 8 位寄存器能分别寻址。这样，可以将数据寄存器当作一个 16 位寄存器，也可用做两个 8 位寄存器。

数据寄存器可以用来存放 8 位或 16 位的二进制操作数，这些操作数可以是参加操作的数据、操作的中间结果，也可以是操作数据的地址。大多数算术和逻辑运算指令可以使用这些数据寄存器。

### （2）指针和变址寄存器

另外 4 个通用寄存器是堆栈指针 **SP**、基址指针 **BP**、源变址寄存器 **SI** 及目的变址寄存器 **DI**。这 4 个 16 位的寄存器只能按 16 位进行存取操作，主要用来形成操作数的地址，用于堆栈操作和变址运算中计算操作数和有效地址。其中 **SP**、**BP** 用于堆栈操作，**SP** 用来确定堆栈在内存中的地址，**BP** 用来存放在现行堆栈段的一个数据区的“基址”。**SI**、**DI** 用于变址操作，存放变址地址。这 4 个寄存器也可用做数据寄存器。

在 8086/8088 的指令系统中，在许多情况下，一条指令的功能只能用一个特定的寄存器或寄存器组来完成；对某些完成特定操作的 8088 指令，上述通用寄存器具有一些隐含用法。

## 2. 指令指针 IP

这是一个 16 位的专用寄存器，**IP** 指向当前需要取出的指令字节，当 **BIU** 从内存中取出一个指令字节后，**IP** 自动加 1，指向下一个字节。**IP** 指向的是指令地址的段内地址偏移量，也称为偏移地址或有效地址。

程序员不能对 **IP** 进行存取操作，程序中的转移指令、返回指令，以及中断处理能对 **IP** 进行操作。

## 3. 标志寄存器 FR 或程序状态字 PSW

8086/8088 有一个 16 位的标志寄存器 **FR**。**FR** 中有 6 个状态位、3 个控制位，7 位未用。所用的 9 位如图 2-3 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

图 2-3 8086/8088 标志寄存器

### （1）状态标志位

① 进位标志 **CF**：反映算术运算后，最高位（字节操作为 **D<sub>7</sub>**，字操作为 **D<sub>15</sub>**）出现进位（或借位）的情况，有则为“1”。**CF** 主要用于字节数的加、减法运算，移位及循环指令也会改变 **CF** 的值。

② 奇偶标志 **PF**：反映操作结果中“1”的个数的情况，若为偶数，则 **PF**=1。主要在数据通信中用来检查数据传送有无出错。

③ 辅助进位标志 **AF**：反映一个 8 位量（16 位量的低位字节）的低 4 位向高 4 位有无进位（或借位）的情况，有则置“1”。**AF** 用于十进制算术运算指令。

④ 零标志 ZF: 反映运算结果是否为零, 结果为零, ZF 置为“1”。

⑤ 符号标志 SF: 反映运算结果的符号情况。若结果为负数, 则 SF 为“1”。SF 的取值与运算结果最高位(字节操作为  $D_7$ , 字操作为  $D_{15}$ )一致。

⑥ 溢出标志 OF: 反映带符号数(二进制补码表示)运算结果是否超过机器所能表示的数值范围的情况。对字节运算为  $-128 \sim +127$ , 对字运算为  $-32768 \sim +32767$ 。若超过上述范围则称为“溢出”, OF=1。

上述 6 个状态标志由 EU 设置, 反映算术或逻辑运算结果的某些状态, 有一组指令使程序能根据这些标志的状态, 即前一次运算的结果而改变其执行的方向。不同的指令对状态标志的影响也不同。

## (2) 控制标志位

① 方向标志 DF: 在进行字符串操作时, 每执行一条串操作指令, 对地址要进行一次调整(对字节操作为加 1 或减 1, 对字操作为加 2 或减 2), 由 DF 决定地址是增还是减。若 DF=1, 则为减量; 若 DF=0 则为增量。

② 中断允许标志 IF: 表示系统是否允许外部的可屏蔽中断。若 IF=1, 则表示允许中断。IF 对非屏蔽中断及内部中断请求不起作用。

③ 跟踪标志 TF: 当 TF=1 时, CPU 每执行完一条指令, 便自动产生一个内部中断, 对程序进行逐条检查, 称为“单步工作方式”, 常用于程序的调试。

上述 3 个控制标志可由指令来设置或清除, 以控制 8086/8088 的操作。

## 4. 段寄存器 SR

计算机系统的内存中通常存放三种信息: ①代码(指令), 指示计算机执行何种操作; ②数据(字符、数值), 是程序处理的对象; ③堆栈信息, 用来保存返回地址和中间结果。

为清晰起见, 这三种信息通常分别存放在各自的存储区域内——8086/8088 存储系统中的不同存储段。8086/8088 系统中把可直接寻址的 1MB 内存空间分成若干个逻辑段, 每个段的物理长度为 64KB。每个段的起始地址的有关值存放在段寄存器的 4 个 16 位寄存器中。这 4 个段寄存器为:

- 代码段寄存器 CS: 指向当前的代码段, 指令由此段取出。
- 堆栈段寄存器 SS: 指向当前的堆栈段, 堆栈操作所需的就是该段存储单元的内容。
- 数据段寄存器 DS: 指向当前数据段, 通常用来存放程序变量。
- 附加段寄存器 ES: 指向当前附加段, 通常也用来存储数据。

8086/8088 利用上述寄存器的内容, 可以访问这 4 个存储区段。

## 2.1.3 8086/8088 的存储器组织及地址形成

### 1. 存储器组织

8086/8088 CPU 有 20 条地址线, 可以配置 1MB 的存储器, 地址编号为  $00000H \sim FFFFFH$ 。存储空间都按字节进行组织, 每个存储单元存储一个字节的数据, 若存放“字”数据(16 位), 则存放在相邻两个存储单元之中, 高字节存放在高地址单元, 低字节存放在低地址单元。指令、字节数据和字数据可以自由地存放在任何字节的地址中, 而不必考虑“对准”问题, 因此代码可以紧凑地存储在存储器中, 从而节省了存储空间。

## 2. 存储器分段

因为 8086/8088 CPU 的内部数据通路和寄存器皆为 16 位，内部 ALU 只能进行 16 位运算，因此 8086/8088 CPU 对地址只能进行 16 位运算。寻址范围仅为  $2^{16}=65536=64\text{KB}$ 。所以需要引入“分段”概念，以获得 20 位地址。

8086/8088 的程序把 1MB 的存储空间看成为一组存储段，其各段的功能由具体用途而定。一个存储段是存储器的一个逻辑单位，其长度可达 64KB，每个段都由连续的存储单元构成，并且是存储器中独立的可分别寻址的单位。每段第一个字节的位置称为“段起始地址”，可由软件指定。对段起始地址的要求是：必须能被 16 整除（即 XXXX0H）。段寄存器中存放了与段起始地址有关的 16 位“段基址”，一旦 4 个段寄存器被赋值后，程序就可访问 4 个段中的任一存储单元。若程序超过 64KB 时，要通过给段寄存器重新送新值，把超出部分转到下一段中去。

注意，几个段可以相互重叠，也可指向同一个 64KB 空间。

## 3. 物理地址的生成

在具有地址变换机构的计算机中，有两种存储器地址：①允许在程序中编排的地址——逻辑地址；②信息在存储器中实际存放的地址——物理地址。在 8086/8088 系统中每个存储单元也都被认为有这两种地址。

8086/8088 与存储器之间的所有信息交换都要使用 20 位的物理地址，而程序中所涉及的地址都是 16 位的逻辑地址，对所给定的任一存储单元而言有两部分逻辑地址：段基址——决定了该段第一个字节的位置；段内偏移量——该存储单元相对于该段起点字节的距离。

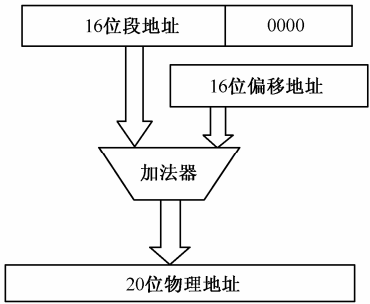


图 2-4 8086/8088 物理地址的形成

段基址存放在段寄存器 CS、SS、DS 和 ES 中；而段内偏移量由 SP、BP、SI、DI、IP，以及上述数据寄存器的组合而组成。

存储单元的 20 位物理地址是通过将 16 位的段基址左移 4 位再加上 16 位的偏移地址而生成的，即：

$$\text{物理地址} = \text{段基址} \times 10\text{H} + \text{段内偏移量}$$

物理地址的计算方法如图 2-4 所示。

例如，6000:0280 的物理地址为 60280H。

8086/8088 CPU 中 BIU 单元的加法器用来完成物理地址的计算。

## 4. 段寄存器的使用

8086/8088 对访问不同的内存段所使用的段寄存器和相应的偏移地址的来源有一些具体的约定，如表 2-1 所示。

表 2-1 对段寄存器使用的约定

	内存访问类型	默认段寄存器	可指定段寄存器	段内偏移地址来源
1	取指令	CS	无	IP
2	堆栈操作	SS	无	SP

续表

	内存访问类型	默认段寄存器	可指定段寄存器	段内偏移地址来源
3	源串	DS	CS、ES、SS	SI
4	目的串	ES	无	DI
5	BP 用做基址寻址	SS	CS、ES、DS	按寻址方式计算得到的有效地址
6	一般数据存取	DS	CS、ES、SS	按寻址方式计算得到的有效地址

2.1.4  8086/8088 的 I/O 端口组织

8086/8088 系统与所有外部设备的连接都是通过 I/O 接口电路进行的。一个外设可能有数据、状态和控制命令寄存器，每个寄存器往往对应一个端口。这样，一个外设可能有一个或几个端口，如何来区分不同外设的不同端口呢？如同存储器中用不同地址来区分存储单元一样，可采用地址来对端口加以区分，给每个端口分配一个地址，此地址就叫做端口号。各个端口号不能重复。

8086/8088 具有专用的 I/O 指令，地址线通过 I/O 端口译码器产生一组 I/O 端口地址，这是与内存地址不同的另一组地址分布。

以 8086/8088 为 CPU 组成的系统中，若采用直接寻址方式，可寻址 256 个端口地址。而利用间接寻址时，最大可寻址 64K 端口地址。在 IBM PC/XT 系统中，用 A<sub>0</sub>~A<sub>9</sub> 的 10 根地址线对 I/O 端口寻址，故最大 I/O 端口地址空间为 1K。

2.2  8086/8088 的外部结构

8088/8086 CPU 都是具有 40 个引脚的集成电路芯片，采用双列直插式封装。图 2-5 所示为 8088 的引脚图，8086 与之基本相同。

为了减少芯片的引线数量，8088 的许多引脚具有双重定义和功能，采用分时复用方式工作，即在不同时刻，这些引脚上的信号是不相同的。同时，8088 的最大和最小两种工作模式可以通过在 MN/MX 输入引脚加上不同的电平来进行选择。

当 MN/MX =1 时，8088 工作在最小模式。此时，构成的微型计算机中只包括一个处理器 8088，且系统总线由 8088 的引线直接引出形成，系统所用的芯片最少。当 MN/MX =0 时，8088 工作在最大模式之下。在此模式下，构成的微型计算机中除了有 8088 CPU 之外，还可以接另外的处理器（如 8087 数学协处理器），构成多微处理器系统。在最大模式下，微机的

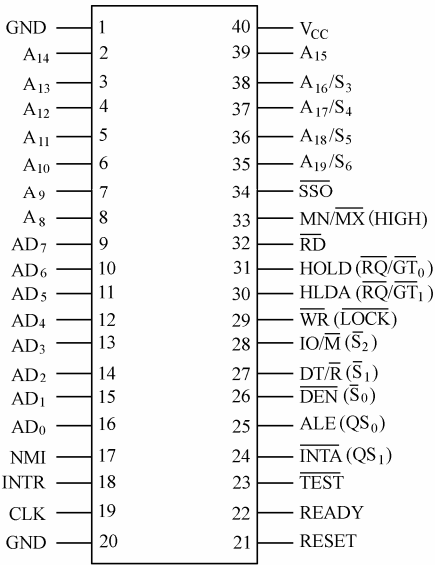


图 2-5  8088 微处理器引脚图



系统总线要由 8088 和总线控制器（8288）共同形成。图 2-5 中括号内的引脚信号用于最大模式。

### 1. 最小模式下的引脚

在最小模式下，8088 的引脚定义为：

- $A_{16} \sim A_{19}/S_3 \sim S_6$ ：地址、状态复用的引脚，三态输出。在 8088 执行指令过程中，某一时刻从这 4 个引脚上送出地址的高 4 位  $A_{16} \sim A_{19}$ ，而在其他时刻，这 4 个引脚送出状态信号  $S_3 \sim S_6$ 。这些状态信息中， $S_6$  恒等于 0， $S_5$  指示中断允许标志位 IF 的状态， $S_4$ 、 $S_3$  的组合指示 CPU 当前正在使用的段寄存器，其编码见表 2-2。

表 2-2 段寄存器编码

$S_4$	$S_3$	当前正在使用的段寄存器
0	0	ES
0	1	SS
1	0	CS 或未使用任何段寄存器
1	1	DS

- $A_{15} \sim A_8$ ：地址输出，三态。CPU 寻址内存或接口时，从这些引脚送出地址  $A_{15} \sim A_8$ 。

- $AD_7 \sim AD_0$ ：地址、数据分时复用的双向信号线，三态。当  $ALE=1$  时，这些引脚上传输的是地址信号。当  $ALE=0$  时，这些引脚上传输的是数据信号。

- $\overline{IO/\overline{M}}$ ：输入输出/存储器控制信号，三态。用来区分当前操作是访问存储器还是访问 I/O 端口。若此引脚输出为低电平，则访问存储器；若输出为高电平，则访问 I/O 端口。

- $\overline{WR}$ ：写信号输出，三态。此引脚输出为低电平时，表示 CPU 正在对存储器或 I/O 端口进行写操作。

- $\overline{DT/\overline{R}}$ ：数据传送方向控制信号，三态。用于确定数据传送的方向。当其为高电平时，CPU 向存储器或 I/O 端口发送数据；当其为低电平时，CPU 从存储器或 I/O 端口接收数据。此信号用于控制总线收发器 74LS245 的传送方向。

- $\overline{DEN}$ ：数据允许信号，三态。该信号有效时，表示数据总线上有有效数据。它在每次访问内存或 I/O 端口，以及在中断响应期间有效。常用做数据总线驱动器的片选信号。

- $ALE$ ：地址锁存允许信号，三态输出，高电平有效。当它为高电平时，表明 CPU 地址线上有有效地址。因此，该信号常作为锁存控制信号将  $A_{19} \sim A_0$  锁存到地址锁存器。

- $\overline{RD}$ ：读选通信号，三态，低电平有效。当其有效时，表示 CPU 正在对存储器或 I/O 端口进行读操作。

- $READY$ ：准备好信号输入引脚，高电平有效。它是由被访问的内存或 I/O 设备发出的响应信号，当其有效时，表示存储器或 I/O 设备已准备好，CPU 可以进行数据传送。若存储器或 I/O 设备没准备好，则使  $READY$  信号为低电平。CPU 在  $T_3$  周期采样  $READY$  信号，若其为低，CPU 自动插入等待周期  $T_w$ （1 个或多个），直到  $READY$  变为高电平后，CPU 才脱离等待状态，完成数据传送过程。

- $INTR$ ：可屏蔽中断请求输入信号，高电平有效。CPU 在每条指令的最后一个周

期采样该信号，以决定是否进入中断响应周期。该引脚上的中断请求信号可用软件屏蔽。

- $\overline{\text{TEST}}$ ：测试信号输入引脚，低电平有效。当 CPU 执行 WAIT 指令时，每隔 5 个时钟周期对此引脚进行一次测试。若为高电平，则 CPU 继续处于空转状态进行等待，直到  $\overline{\text{TEST}}$  引脚变为低电平，CPU 才结束等待状态，继续执行下一条指令。

- NMI：非屏蔽中断请求输入信号，上升沿触发。该引脚上的中断请求信号不能用软件屏蔽，CPU 在当前指令执行结束后就进入中断过程。

- RESET：系统复位输入信号，高电平有效。为使 CPU 完成内部复位过程，该信号至少要在 4 个时钟周期内保持有效。复位后 CPU 内部寄存器的状态如表 2-3 所示。当 RESET 返回低电平时，CPU 将重新启动。

- $\overline{\text{INTA}}$ ：中断响应信号输出，低电平有效，是 CPU 对中断请求信号 INTR 的响应。在响应过程中，CPU 在  $\overline{\text{INTA}}$  引脚连续送出两个负脉冲，可用做外部中断源的中断向量码的读选通信号。

- HOLD：总线保持/请求信号输入，高电平有效。当某一总线上主控设备要占用系统总线时，通过此引脚向 CPU 提出请求。

- HLDA：总线保持/响应信号输出，高电平有效。这是 CPU 对 HOLD 请求的响应信号。当 CPU 收到有效的 HOLD 信号后，就会对其做出响应：使 CPU 的所有三态输出的地址信号、数据信号和相应的控制信号变为高阻状态（浮动状态）；同时还输出一个有效的 HLDA，表示处理器现在已放弃对总线的控制。当 CPU 检测到 HOLD 信号变低后，就立即使 HLDA 变低，同时恢复对总线的控制。

- $\overline{\text{SSO}}$ ：系统状态信号输出。它与  $\overline{\text{IO}/\overline{\text{M}}}$  和  $\overline{\text{DT}/\overline{\text{R}}}$  信号决定了最小模式下当前总线周期的状态。三者的组合所表示的处理器操作见表 2-4。

- CLK：时钟信号输入引脚。8088 的标准时钟频率为 4.77MHz，时钟的占空比为 33%。

- $V_{\text{CC}}$ ：5V 电源输入引脚。

- GND：地线。

表 2-3 复位后的内部寄存器状态

内部寄存器	内容
CS	FFFFH
DS	0000H
SS	0000H
ES	0000H
IP	0000H
FLAGS	0000H
其余寄存器	0000H
指令队列	空

表 2-4  $\overline{\text{SSO}}$ 、 $\overline{\text{IO}/\overline{\text{M}}}$ 、 $\overline{\text{DT}/\overline{\text{R}}}$  的组合及其所对应的操作

$\overline{\text{IO}/\overline{\text{M}}}$	$\overline{\text{DT}/\overline{\text{R}}}$	$\overline{\text{SSO}}$	操 作
1	0	0	发中断响应信号
1	0	1	读 I/O 端口
1	1	0	写 I/O 端口
1	1	1	暂停
0	0	0	取指令
0	0	1	读内存
0	1	0	写内存
0	1	1	无作用

## 2. 最大模式下的引脚

当  $\overline{MN}/\overline{MX}$  引脚加上低电平时，8088 CPU 工作在最大模式之下。此时，除引脚 24~34 外，其他引脚与最小模式完全相同。

- $\overline{S_2}$ 、 $\overline{S_1}$ 、 $\overline{S_0}$ ：总线周期状态信号输出，低电平有效，三态。这 3 个信号连接到总线控制器 8288 的输入端，8288 对它们译码后可以产生系统总线所需要的各种控制信号。 $\overline{S_2}$ 、 $\overline{S_1}$ 、 $\overline{S_0}$  的代码组合以及对应的操作见表 2-5。

表 2-5  $\overline{S_2}$ 、 $\overline{S_1}$ 、 $\overline{S_0}$  的组合及其所对应的操作

$\overline{S_2}$ 、 $\overline{S_1}$ 、 $\overline{S_0}$	对应的操作	$\overline{S_2}$ 、 $\overline{S_1}$ 、 $\overline{S_0}$	对应的操作
0 0 0	发中断响应信号	1 0 0	取指令
0 0 1	读 I/O 端口	1 0 1	读内存
0 1 0	写 I/O 端口	1 1 0	写内存
0 1 1	暂停	1 1 1	无作用

- $\overline{RQ}/\overline{GT_1}$ 、 $\overline{RQ}/\overline{GT_0}$ ：总线请求/总线响应信号引脚。每一个引脚都具有双向功能，既是总线请求输入也是总线响应输出。但是  $\overline{RQ}/\overline{GT_0}$  比  $\overline{RQ}/\overline{GT_1}$  具有更高的优先权。这些引脚的内部都有上拉电阻，所以在未使用时可以悬空。

这两个引脚的功能如下：当其他的总线控制设备要使用系统总线时，就会产生一个总线请求信号（一个时钟周期宽的负脉冲），并把它送到  $\overline{RQ}/\overline{GT}$  引脚，类似于最小模式下的 HOLD 信号。CPU 检测到总线请求信号后，在下一个  $T_4$  或  $T_3$  期间（ $T_4$ 、 $T_3$  为时钟周期），在  $\overline{RQ}/\overline{GT}$  引脚送出总线响应信号（一个时钟周期宽的负脉冲）给请求总线的设备，它类似于最小模式的 HLDA 信号。然后从下一个时钟周期开始，CPU 释放总线。总线请求设备使用完总线后，再产生一个  $\overline{RQ}/\overline{GT}$  信号。CPU 检测到该信号后，从下一个时钟周期开始重新控制总线。

- $\overline{LOCK}$ ：总线封锁信号输出，低电平有效。该信号有效时，CPU 锁定总线，不允许其他的总线控制设备申请使用系统总线。 $\overline{LOCK}$  信号由前缀指令“LOCK”产生，LOCK 指令后面的一条指令执行完后， $\overline{LOCK}$  信号失效。

- $QS_1$ 、 $QS_0$ ：指令队列状态输出。根据该状态信号，从外部可以跟踪 CPU 内部的指令队列。 $QS_1$ 、 $QS_0$  的编码如表 2-6 所示。

表 2-6  $QS_1$ 、 $QS_0$  的组合及其所对应的操作

$QS_1$ 、 $QS_0$	对应的操作
0 0	无操作
0 1	队列中操作码的第一个字节
1 0	队列空
1 1	队列中非第一个操作码字节

- HIGH：在最大模式下始终为高电平输出。

此外，在最大模式下， $\overline{RD}$  引脚不再使用。

## 2.3 8088 的工作模式

在设计 8088 CPU 时, 已经考虑使其能在各种不同用途中工作。根据所构成的计算机系统的复杂程度, 规定了两种工作模式, 即最小模式和最大模式。

最小模式是指构成系统的规模比较小, 只含有 8088 一个处理器, 三大总线连接比较简单。系统的地址总线除了  $A_{15} \sim A_8$  直接由 CPU 的  $A_{15} \sim A_8$  提供外, 其余 12 根地址线则由 CPU 的  $A_{19} \sim A_{16}$ 、 $AD_7 \sim AD_0$  通过地址锁存器 8282 提供。系统的数据线可由 CPU 的  $AD_7 \sim AD_0$  直接提供, 也可以通过收发器接口芯片 8286 提供, 以增强数据的驱动能力, 这视负载情况而定。系统的控制总线直接由 CPU 的控制总线提供, 这样系统中与总线控制有关的逻辑电路最简单。

其外部配置如图 2-6 所示。

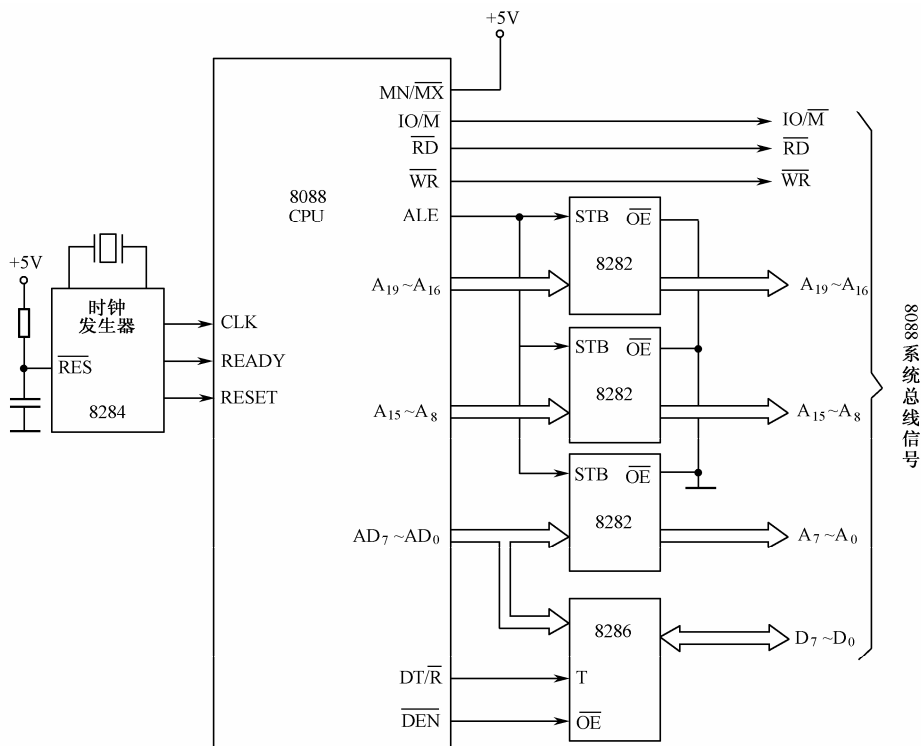


图 2-6 8088 最小模式下的外部配置

最大模式是指除了 8088 CPU 之外, 可能还含有一片或多片微处理器。8088 主处理器之外的其他处理器, 称为协处理器, 如数值运算器 8087 和外设 I/O 处理器 8089。另外一种情况是, 当系统规模较大时, 即使是单 CPU, 也组成最大模式。此时系统中加有一个总线控制器 8288, 以提高驱动能力, 提供较好的大系统性能。例如, IBM PC/XT 就属于这种情况, 它在主板上, 为 8087 预留一个空着的 40 芯插座, 并且通过 I/O 扩展槽, 很容易扩展成多处理器系统。其外部配置如图 2-7 所示。

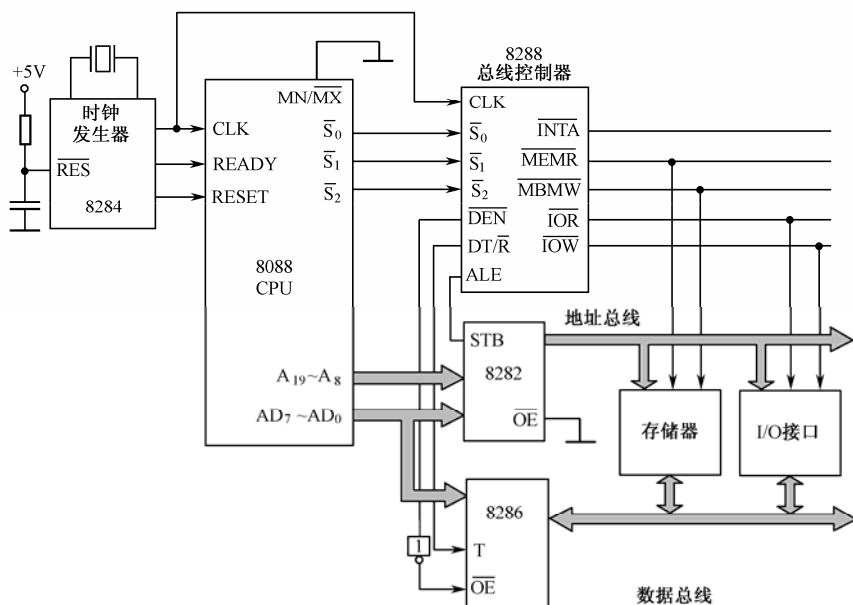


图 2-7 8088 最大模式下的外部配置

## 2.4 8086/8088 的总线操作和时序

微处理器是按照一定的时序来工作的。时序有两种，即时钟周期和总线周期。一条指令的执行需要若干个总线周期才能完成，而一个总线周期又由若干个时钟周期构成。

微处理器在运行过程中按照一个统一的时钟一步步地执行每一个操作。每个时钟脉冲的持续时间就称为一个时钟周期。显然，时钟周期越短，CPU 执行的速度就越快。

在 8088 CPU 中，CPU 与内存或端口之间都是通过总线来进行通信的，如将一个字节写入内存单元中，或者从内存中的某单元读一个字节到 CPU。这种通过总线进行一次读或写的过程称为一个总线周期。一个总线周期包括多个时钟周期。典型的总线周期如图 2-8 所示。

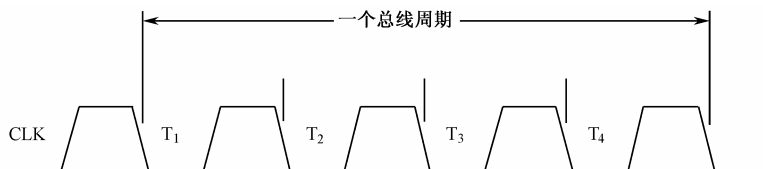


图 2-8 典型的总线周期

由图 2-8 可知, 8088 CPU 的一个总线周期通常包含 4 个 T 状态: T<sub>1</sub>、T<sub>2</sub>、T<sub>3</sub> 和 T<sub>4</sub>。所谓一个 T 状态就是一个时钟周期, 它是 CPU 执行操作的最小时间单位, 如 8088 的时钟频率为 5MHz, 则一个 T 状态为 200ns。

对于 8086/8088 来说,基本的总线周期有如下三种:①存储器读或写总线周期;②外设端口的读或写总线周期;③中断响应总线周期。

下面简要介绍 8088 CPU 在最小模式下的读/写时序。最大模式下的时序除有些信号是由总线控制器 (8288) 产生的以外, 其基本时间关系与最小模式大致相同。

8088 读总线周期和 8088 写总线周期分别如图 2-9 和图 2-10 所示。

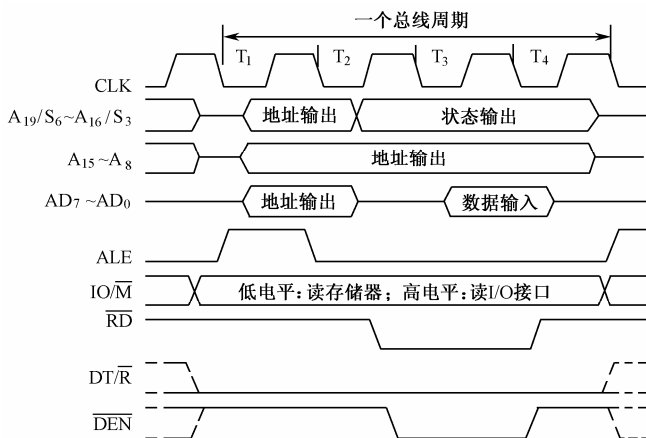


图 2-9 8088 读总线周期

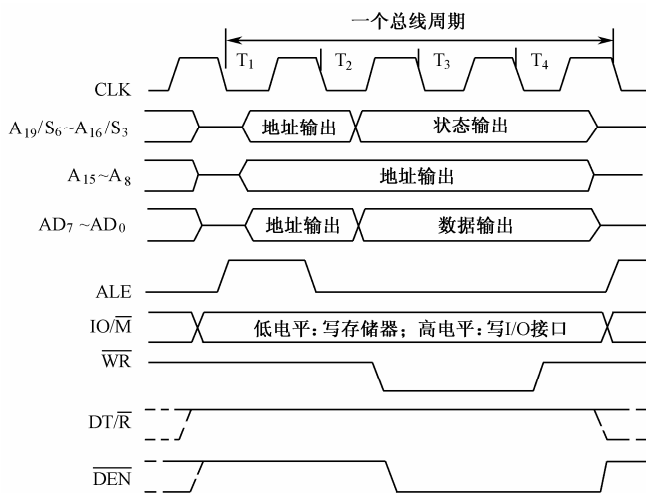


图 2-10 8088 写总线周期

由图可知, 正常的 8088 总线周期, 不管是读还是写, 都由至少 4 个时钟周期 ( $T_1 \sim T_4$ ) 组成。在  $T_1$  期间, 地址信号线  $A_{15} \sim A_8$ 、地址/状态复用信号线  $A_{19}/S_6 \sim A_{16}/S_3$  和地址/数据复用信号线  $AD_7 \sim AD_0$  分别送出地址  $A_{15} \sim A_8$ 、 $A_{19} \sim A_{16}$  和  $A_7 \sim A_0$ , 与此同时送出地址锁存允许信号 ALE。外部电路利用 ALE 把这些地址信号锁存到地址锁存器中, 即可在锁存器的输出端得到完整的 20 位地址信号  $A_{19} \sim A_0$ 。一旦 ALE 的高电平将地址信号锁存, 在此后的时钟周期中, 即可利用有关的控制信号完成对内存或外设的读/写操作。在写总线周期中, CPU 从  $T_2$  开始把数据送到总线上并维持至  $T_4$ 。在读总线周期中, CPU 在  $T_3$  到  $T_4$  期间读入总线上的数据。

在某些情况下，当内存或端口的速度比较慢，使得在 4 个时钟周期里不能完成读/写操作时，可通过时钟发生器（8284）产生一个低电平信号送到 8088 的 READY 端。8088 CPU 在每个总线周期的  $T_3$  的开始处都要检查 READY 的状态。若此时 READY 为低电平，则 CPU 不执行  $T_4$ ，而是在  $T_3$  之后插入一个等待时钟周期  $T_w$ （图中未画出），以等待存储器或 I/O 端口完成读/写操作。在  $T_w$  的开始时刻，CPU 还要检查 READY 状态，若仍为低电平，则再插入一个  $T_w$ 。此过程一直进行到某个  $T_w$  开始时，READY 已经变为高电平，这时下一个时钟周期就是总线周期的最后一个时钟周期了。由此可见，利用 READY 信号，CPU 可以插入若干个  $T_w$ ，使总线周期延长，达到可靠地读/写内存和 I/O 端口的目的。

另外还要注意一点，CPU 的读或写是在  $T_4$  开始时刻（或读/写信号的后沿）进行的，这时数据线上的数据已经达到稳定状态，只有这样，利用 READY 插入  $T_w$  周期才有意义。

## 2.5 8086/8088 的横向提升

8086 CPU 作为一种高性能的 16 位微处理器，其处理能力比 8 位的 8080 CPU 提高了 10 倍以上，其寻址能力、指令功能也大为增强。但由于 8086 自身功能有限以及扩大应用的需要，Intel 公司在 8086 的基础上，对它进行了横向和纵向性能的提升，从而形成了一个完整的 16 位微机系列。

在 8086/8088 系列中，把单纯只要 8086 CPU 的，命名为 iAPX86/10，若配上各种协处理器，则性能横向提升。与数值数据协处理器 8087 配接，成为 iAPX86/20；再与输入/输出协处理器 8089 配接后，就成为 iAPX86/21；与操作系统固件 80130 配接后，就成为 iAPX86/30。若从 8086 自身的基础结构加以改进，提高其运算速度、功能，增加存储器管理和虚地址保护机构，则使 8086 的性能得到纵向提升。为此，Intel 公司又推出了超级 16 位的 80286，形成 iAPX286。

### 2.5.1 数值数据协处理器 8087

8087 是一种专门为提高系统处理数值数据运算能力而设计的协处理器，它不仅能实现多种数据类型的高精度数值运算，还可以进行一些超越函数的计算。

8087 的内部结构可分为两大部分：控制单元（CU）和数值处理单元（NEU）。CU 保持与 CPU 同步操作，它利用 CPU 发出的状态信息（ $S_2 \sim S_0$ ）及时了解其正在进行什么操作，一旦 CPU 取指令，CU 就与 CPU 一起并行对指令译码，从中识别并取得由 8087 执行的指令。CU 执行控制类指令，并读/写存储器，以获得操作数和传送运算结果。NEU 中设置有指令队列，而且与 8086/8088 CPU 中的指令队列一致；由 CU 通过 BHE 的状态来检测 8087 所配合的主 CPU 是 8086 还是 8088。当 BHE=0 时，表示 CPU 是 8086；而当 BHE=1 时，表示 CPU 是 8088。CU 将设置与 CPU 相同长度的指令队列。8087 执行所有的数值运算指令，并以交权指令（ESC）的形式给出。当 CU 得到一条 ESC 指令时，会根据指令的类型确定是应该由 CU 本身完成的控制类指令，还是应该由 NEU 执行的数值运算指令。NEU 完成的运算指令包括算术运算、逻辑运算、超越函数运算及常数类指令。NEU 内部数据宽度可达到 80 位（64 位尾数，15 位阶码和 1 位符号位），因此，其数据传送速率很高。一旦

NEU 开始执行指令，就立即将 BUSY 置“1”，送到 CPU 的 TEST 端，CPU 利用 WAIT 指令可查询 TEST 的状态，了解当前 8087 所处的状态。

8087 的内部寄存器组包括 8 个 80 位字长的数据寄存器  $R_7 \sim R_0$ ，它们构成一个寄存器堆栈，也是按“后进先出”的原则来完成压栈和弹出操作的，但其操作方式与存储器中的堆栈不同，它被称为“硬堆栈”，而存储器中的堆栈被称为“软堆栈”。另外，有 4 个专用寄存器，即：16 位控制寄存器 CR，16 位状态寄存器 SR，32 位指令指示器 IP 和 32 位数据指示器 DP。它们为浮点数据类型、整数数据类型、寄存器及指令系统提供了有力的硬件支持。

8087 芯片的封装外形与 8086 基本相同，有 40 条双列直插式引脚。其地址/数据、状态、准备好、复位、时钟、接地和电源等引脚与 8086/8088 的引脚位置相同，其余 8 条引脚中，有 4 条（17, 18, 29, 30）未用。其他引脚是：BUSY（23）引脚接至 8086 CPU 的 TEST（23），用于检测当前 8087 的忙闲状态； $\overline{RQ}/\overline{GT}_0$ （31）引脚接至 8086 CPU 的  $\overline{RQ}/\overline{GT}_0$  或  $\overline{RQ}/\overline{GT}_1$ ，以传送总线请求和允许信号；INT（32）引脚接至中断管理逻辑（假设允许 8087 中断），通常与中断控制器 8259A 相连，用来向 CPU 发出中断请求； $\overline{RQ}/\overline{GT}_1$ （33）可以接至独立处理机（如 8089）的请求/允许引脚。这种接口比较简单，只需用 8086/8088 和 8087 组合的复合模块代替原来的 CPU，就能很容易地改进现有的最大方式 8086/8088 系统的性能。

8087 的指令系统共包括 69 条指令，按功能可分为数据传送、算术运算、比较、超越函数计算、取常数和处理器控制等 6 大类指令。8087 作为专用于数值运算的协处理器，不仅提供了各种形式的高精度的加、减、乘、除运算指令，还提供了求平方根、绝对值、指数、正切等指令。采用最大方式工作的多处理器系统，比 8086 CPU 的系统在数学运算能力方面提高了 100 倍左右。此外，也弥补了它所缺少的双倍字长以上的各种算术运算功能。由于增加了 32 位、64 位、80 位浮点运算和 18 位 BCD 码数据运算的指令，其运算速度之快和运算种类之多都远远超过其他 16 位微处理器。

### 2.5.2 输入/输出协处理器 8089

8089 是一种专门为提高系统输入/输出处理能力而设计的协处理器，它可方便地将 8086/8088 CPU 与 8/16 位的外部设备连接起来相互通信。它具有自己的指令系统，能执行程序，除了可完成输入/输出操作外，还具有对传送的数据进行装配、拆卸、变换、校验或比较等多种功能，从而可大大减轻 CPU 在输入/输出处理过程中的开销，有效地提高系统的性能。

当系统中设置了 8089 以后，8086/8088 CPU 必须以最大方式工作。当 CPU 需要进行输入/输出操作时，只需在存储器中建立一个信息块，设置好需要执行的操作和有关参数，然后通知 8089 去读取这些信息，8089 读取操作控制信息后，即可按 DMA 方式自动地完成全部的输入/输出操作。如果在数据输入/输出过程中出现差错，8089 还可进行重复传送或必要的处理。对配有 8089 的 CPU 来说，所有的输入/输出操作中，数据都是整块地成批发送或接收的。例如，和 CRT 终端、行式打印机进行的按字或字节的数据传送都由 8089 来完成。在整个数据块的输入/输出过程中，CPU 不必干预，可并行地执行其他操作。

8089 与 8086/8088 CPU 协同工作时，有两种基本的结构方式。一是本地方式。在这种方式下，8089 与 8086/8088 CPU 共享系统总线和 I/O 总线，可在不增设其他硬件的情况下



完成两个 DMA 通道的功能。这时，8086/8088 CPU 是系统总线和 I/O 总线的主控者，而 8089 是 CPU 的从属设备。当需要使用总线时，8089 可向 CPU 申请总线使用权。CPU 响应这一请求后，可将总线使用权授予 8089。一旦 8089 使用完总线，将自动放弃总线使用权，然后才能由 CPU 重新接管总线。二是远程方式。这是一种高效率的工作方式，在这种方式下，8089 与 CPU 之间仍然共享系统总线，但 8089 还具有它自己的局部 I/O 总线。由于系统总线与局部 I/O 总线可并行操作，因此可大大提高 8089 与 CPU 之间并行工作的程度。当 CPU 使用系统总线时，8089 可通过自己的局部 I/O 总线访问 I/O 设备和本地存储器，只要要求其存储容量小于 64KB，且不允许 CPU 访问。

8089 的内部结构由公共控制单元（CCU）、算术逻辑运算单元（ALU）、装配/拆卸寄存器、取指令部件、总线接口单元（BIU）和两个独立的“智能”型 DMA 通道组成。每个通道都有自己的寄存器组和 I/O 控制器。I/O 控制器用来管理通道的 DMA 操作和向 CPU 发出中断请求。两个通道可根据各自的优先级交替地工作。通道传输速度最高可达 1.25MB/s。8089 除能完成一般的输入/输出操作之外，还具有执行程序的能力，所执行的程序称为通道程序（即源程序模块）。通道程序是用汇编语言 ASM-89 编写的，经汇编后得到一个可浮动的目标程序模块，此目标程序模块交由 8089 即可执行。

## 2.6 80x86 高档微处理器

### 2.6.1 80286 的体系结构

Intel 公司在 20 世纪 80 年代初推出了新的微处理器——Intel 80286，这是一种具有存储器管理和保护机构的 16 位微处理器。80286 芯片采用 68 个引脚四列直插式封装，时钟频率有 8MHz 和 10MHz 两种。

#### 1. 80286 的内部结构

80286 在内部结构上比 8086 增加了指令执行部件（IU），同时将 8086 中的总线接口部件分成总线部件（BU）、地址部件（AU）。这样，80286 的 CPU 有四个独立的处理部件：EU、AU、BU、IU，这四个部件并行操作。

#### 2. 80286 的寄存器组

80286 具有 15 个 16 位的寄存器，可以分成三组：通用寄存器、段寄存器、状态和控制寄存器。其中通用寄存器和段寄存器与 8086 完全一样，而状态和控制寄存器中有 3 个专用寄存器，用来记录或控制 80286 的某些状态，包括状态标志寄存器（F）、指令指示器（IP）、机器状态字寄存器（MSW）。

##### （1）指令指示器 IP

指令指示器 IP 为 16 位的寄存器，用来指出下一条要执行的指令的偏移地址。

##### （2）状态标志寄存器 F

状态标志寄存器 F 用来记录算术或逻辑运算类指令操作结果的性质。在进行 I/O 操作时，I/O 所在的特权级，也由 F 寄存器来记录。

I/O 特权标志 IOPL（第 12、13 位）：用来指定 I/O 操作时处于 0~3 特权级中的哪一级。

嵌套任务标志 NT（第 14 位）：用来表示当前执行的任务是否嵌套在另一个任务中。当

NT=1 时,表示当前任务被嵌套于另一个任务中,执行完该任务后,要返回到原来的任务中去。

### (3) 机器状态字寄存器 MSW

MSW 用来表示当前处理器所处的状态。目前只使用了它的低 4 位,其中一位用来使 CPU 进入虚地址保护方式,其他三位则起控制协处理器接口作用,具体功能如表 2-7 所示。其中,PE 为保护方式允许位,用来启动微处理器工作方式:PE=0,为实地址方式;PE=1,为虚地址方式。

表 2-7 机器状态字寄存器功能表

位	名称	功 能
0	PE	保护方式允许,把 80286 置于保护方式,并且除 RESET 外,不能被消除
1	MP	监督协处理器,允许 WAIT 指令引起“协处理器不存在异常”
2	EM	仿真协处理器,当 ESC 指令允许仿真一个协处理器时,将引起“协处理器不存在异常”
3	TS	任务转换,表示下一条若使用协处理器指令时,将会引起异常,允许用软件测试当前协处理器处理的上、下文是否属于当前任务

当执行 RESET 后,MSW 被自动置成 FFF0H,即将 80286 置成实地址方式。用 LMSW 和 SMSW 指令可在实地址方式装入和存储机器状态字存储器的内容。

### 3. 80286 的外部引脚

80286 的地址总线 and 数据总线不采用分时复用,而是独立地设置了 24 根地址线和 16 根数据线。下面分别介绍各引脚的功能。

(1) 数据总线  $D_{15} \sim D_0$  (输入/输出,三态)。高电平有效,在存储器、I/O 或中断响应读周期时输入数据;在存储器、I/O 写周期时输出数据。

(2) 地址总线  $A_{23} \sim A_0$  (输出,三态)。高电平有效,输出存储器或 I/O 端口地址。在 I/O 传送时, $A_{23} \sim A_{16}$  为低电平。当数据在低字节中传送时, $A_0$  为低电平。

(3) 系统总线 CLK (输入)。为 80826 提供定时脉冲。通常是一个 16MHz 的信号,在 80826 内部经二分频变为 8MHz 的处理器时钟。

(4) 总线高位允许  $\overline{HEB}$  (输出,三态)。低电平有效,在读、写或中断响应周期,把一个字节传送到数据总线的高 8 位。

(5) 总线周期状态  $\overline{S_1}$ 、 $\overline{S_0}$  (输出,三态)。低电平有效,这两个信号的变化,表示开始一个总线周期,并与  $M/\overline{IO}$  和  $COD/\overline{INYTA}$  一起决定总线周期的类型。

(6) 存储器或 I/O 选择  $M/\overline{IO}$  (输出,三态)。用于区别是访问存储器还是访问 I/O,若为高电平,则访问存储器;若为低电平,则访问 I/O。

(7) 代码/中断响应  $COD/\overline{INYTA}$  (输出,三态)。在存储器操作时,用来区分取指令还是读数据;在 I/O 操作时,用来区分是读 I/O 还是中断响应。80286 总线周期状态定义如表 2-8 所示。

(8) 总线封锁  $\overline{LOCK}$  (输出,三态)。低电平有效,表示在当前的总线周期后,其他主设备不能取得系统总线的控制权。

(9) 总线装备就绪  $\overline{READY}$  (输入)。用来结束一个总线周期,低电平有效。

表 2-8 80286 总线周期状态定义

COD/ $\overline{\text{INYTA}}$	M/ $\overline{\text{IO}}$	$\overline{\text{S}}_1$	$\overline{\text{S}}_0$	启动的总线周期
0	0	0	0	中断响应
0	0	0	1	保留
0	0	1	0	保留
0	0	1	1	非状态周期
0	1	0	0	若 A=1, 则暂停, 否则停机
0	1	0	1	读存储器数据
0	1	1	0	写存储器数据
0	1	1	1	非状态周期
1	0	0	0	保留
1	0	0	1	读 I/O
1	0	1	0	写 I/O
1	0	1	1	非状态周期
1	1	0	0	保留
1	1	0	1	读存储器指令
1	1	1	0	保留
1	1	1	1	非状态周期

(10) 总线请求 **HOLD** 和响应 **HLDA**。**HOLD** 是当其他设备占用总线时, 向 CPU 发出的总线请求信号, 高电平有效。当 CPU 接收到 **HOLD** 后, 输出一个高电平有效的总线响应信号 **HLDA**, 同时 CPU 让出总线使用权。当 CPU 检测出 **HOLD** 信号由高电平变为低电平后, 则使 **HLDA** 变为低电平, 同时 CPU 又获得总线使用权。

(11) 可屏蔽中断请求 **INTR** 和不可屏蔽中断请求 **NMI** (输入)。**INTR** 是一个电平输入信号, 高电平有效; **NMI** 是用上升沿触发的, 这个信号不能用软件加以屏蔽。

(12) 协处理操作请求 **PEREQ** 和响应 **PEACK**。把 80286 的存储器管理和保护能力扩展到协处理器。**PEREQ** 高电平有效, 当输入时, 请求 80286 为协处理器执行一个数据操作数的传送。当请求的操作数开始传送时, **PEACK** 输出一个通知协处理器新事件, 低电平有效。

(13) 协处理器忙 **BUSY** 和出错 **ERROR** (输入)。用来指示 80286 的协处理器操作情况。当 **BUSY** 有效时, 使 80286 程序的执行按照 **WAIT** 和一些 **ESC** 指令的方式停下来, 直到 **BUSY** 变为无效 (高电平)。在等待 **BUSY** 变为无效期间, 8286 可以被中断。当 **ERROR** 有效 (低电平), 使 80286 在执行 **WAIT** 和一些 **ESC** 指令时, 实现协处理器的中断。

(14) 系统复位 **RESET** (输入)。高电平有效。80286 在任何时候都可以用 **RESET** 引脚上的一个由低电平变为高电平的正跳变来初始化, 这个高电平至少要保持 16 个系统时钟周期。

#### 4. 80286 的实地址与虚地址保护方式

##### (1) 80286 的实地址方式

80286 的实地址方式与 8086 工作方式基本相同。为了和 8086 兼容, 80286 的 24 根地址线中只有低 20 位 ( $A_{19} \sim A_0$ ) 有用, 因此, 在实地址方式下可寻址空间为 1MB。在实地址方式中, 80286 保留了两个固定的存储区域。在存储器高端  $0\text{FFFF}0\text{H} \sim 0\text{FFFFFH}$ , 保留的

是系统初始化区；在存储器低端 000000H~0003FFH，存放的是中断矢量表。

### (2) 80286 虚地址保护方式

80286 在实地址方式下操作只相当于一个快速的 8086，而在虚地址保护方式下操作才能充分发挥 80286 的作用。

在虚地址保护方式中，80286 将实地址方式的功能和存储器管理、对虚拟存储器的支持，以及对地址空间的保护合为一体，从而使 80286 能可靠地支持多用户系统。在虚地址保护方式中，80286 的 24 根地址线  $A_{23} \sim A_0$  全被用上，因此，可直接寻址的地址空间为 16MB。80286 通过集成在片内的保护机构，能给每个任务提供最大可达 1000MB 的虚拟存储空间。

虚地址保护方式的物理存储器也是由两部分组成的，即段基地址和段内偏移量。段基地址是 24 位的，将段基地址及相应的特征集合在一起，形成一张表——描述符表，存放在存储器的某一区域。于是，在虚地址保护方式下各段寄存器中的内容，不再是段基地址而是一个参数，用这个参数从描述符表中取出相应的描述符，就找到了段基地址，与 16 位偏移量相加就形成所要寻址单元的物理地址。

## 2.6.2 80386 的体系结构

Intel 公司于 1984 年底推出高性能的 32 位微处理器——Intel 80386，它是 80286 的扩充，从而形成了由 8086、80186、80286、80386 组成的完整的 86 系列微处理器。

80386 的最大特点是在 CPU 芯片上集成了一个存储器管理部件 (MMU)，可对  $2^{46}$  的虚拟存储空间和 4000MB ( $2^{32}$ ) 的物理存储空间进行分段和分页管理，段的最大空间为 4000MB。80386 的时钟频率有 16MHz 和 20MHz 两种，每秒钟可持续执行三四百万条指令，性能为 80286 的三倍，已超过许多超级小型机的速度。

### 1. 80386 的内部结构

80386 逻辑上由六个功能部件组成：总线接口部件、代码预取部件、指令译码部件、存储器管理部件、执行部件及控制部件。这六个功能部件采用流水线结构，可以同时处理多条指令，以缩短程序实际执行时间。

总线接口部件在总线周期内对必要的信号线进行控制。在其他五个部件没有传送要求时，总线接口部件执行从存储器预取指令的操作。

代码预取部件从存储器中以 4 个字节为单位预先取出指令，存放在 16 个字节的指令预取队列中。

指令译码部件从代码预取部件中的预取队列里按顺序取出指令并译码。

执行部件与别的部件协同完成指令的功能。

存储器管理部件由分段部件和分页部件构成。分段部件将逻辑地址转换为线性地址，芯片上有一个段描述符高速缓冲寄存器，其中存有当前段的段描述符，它可加快这种转换。分页部件将线性地址转换为物理地址，在芯片中还有页描述符高速缓冲寄存器，它存放着页描述符。分页部件把物理总线地址接到总线接口部件，以执行存储器访问或 I/O 访问。

### 2. 80386 寄存器组

80386 有以下七组寄存器。

### （1）通用寄存器组

80386 有 8 个 32 位的通用寄存器：EAX、EBX、ECX、EDX、ESP、EBP、ESI、EDI。为了与 8086 系列微处理器兼容，各寄存器的低 16 位部分可作为 16 位寄存器使用，这时分别指定为 AX、BX、CX、DX、SP、BP、SI 和 DI。前四个寄存器的低 16 位又可分为高 8 位和低 8 位，作为 8 位寄存器使用。

### （2）段寄存器组

80386 有 6 个 16 位的段寄存器：CS、DS、SS、ES、FS 和 GS。其中 CS 为代码段、SS 为堆栈段、其余为数据段。段寄存器主要用于在实地址方式时存放段基地址；在保护方式时，它作为保存段描述符的选择器。

### （3）专用寄存器组

80386 有两个 32 位的专用寄存器：指令指针寄存器（EIP）和标志寄存器（EFLAGS）。

EIP 的低 16 位称为 IP，用于执行 8086 的指令。

EFLAGS 的低 16 位和 80286 的状态标志寄存器完全相同，新增加的两个标志位在高 16 位中。其中 VM 用于控制方式转移，当 VM=1 时，从保护方式转换到虚拟 8086 方式；当 VM=0 时，恢复保护方式。RF 是恢复标志，当指令执行结束时，RF=0；执行过程中发生中断时，RF=1。因此，在页变换后需检查 RF 标志，若 RF=1，则再执行该指令；若 RF=0，则从下一条指令开始执行。

### （4）控制寄存器组

80386 有 4 个 32 位的控制寄存器：CR<sub>0</sub>~CR<sub>3</sub>。

CR<sub>0</sub> 是机器状态寄存器，各位的功能如表 2-9 所示。PG 为分页允许位，指示是否使用分页。ET、EM、MP 位控制了与协处理器的接口，TS 进行任务切换。PG、PE 的组合用于设置操作方式，以控制 80386 的工作方式，见表 2-10。

表 2-9 CR<sub>0</sub> 各位的功能

位	功 能
PG	规定页的功能。当 PG=1 时，允许分页；当 PG=0 时，禁止分页
ET	规定协处理器的类型：当 ET=1，连接 80387；当 ET=0，连接 80287
TS	规定任务转换。当 TS=1 时，发生任务转换，若执行 ESC 指令，则产生异常 7（协处理器不存在）。当 TS=0 时，不发生任务转换
EM	模拟协处理器。当 EM=1 时，协处理器不存在（异常 7）；当 EM=0 时，与协处理器通信
MP	监视协处理器。MP 和 TS 一起使用，当 MP=1，TS=1 时，执行 ESC 指令或 WAIT 指令，发生异常 7
PE	保护允许。当 PE=1 时，CPU 进入保护方式；当 PE=0 时，CPU 进入实地址方式

表 2-10 由 PG 位和 PE 位设定操作方式

PG	PE	操 作 方 式
0	0	实地址方式
0	1	保护方式（不分页）
1	0	未使用
1	1	保护方式（分页）

CR<sub>1</sub> 是 Intel 公司的保留寄存器。

CR<sub>2</sub> 是页故障线性地址寄存器, 保存最后发生页故障的线性地址。

CR<sub>3</sub> 是页目录地址寄存器, 用来保存页表的基地址。

#### (5) 系统地址寄存器组

80386 有 4 个系统地址寄存器: GDTR、IDTR、LDTR 和 TR。

GDTR 为全局描述符表寄存器, 用来保存 GDT 的 32 位线性基地址和 16 位界限值。

IDTR 为中断描述符表寄存器, 用来保存 IDT 的 32 位线性基地址和 16 位界限值。LDTR 为局部描述符表寄存器, 用来保存 LDT 的 16 位选择器的值。TR 为任务状态寄存器, 用来保存 TS 的 16 位选择器的值。这四个寄存器在保护方式时都可使用, 但在实地址时只能访问 GDTR 和 IDTR。

#### (6) 调试寄存器组

80386 有 8 个 32 位的调试寄存器 DR<sub>0</sub>~DR<sub>7</sub>, 用于调试功能。DR<sub>0</sub>~DR<sub>3</sub> 设定 4 个断点线性地址。DR<sub>6</sub> 是调试状态寄存器, 保存断点的当前状态; DR<sub>7</sub> 是调试控制寄存器, 设置断点并指示中断结果; DR<sub>4</sub> 和 DR<sub>5</sub> 保留, 供 Intel 公司使用。

### 3. 80386 的外部引线

(1) 时钟 CLK2 (输入)。为 80386 提供定时脉冲, 此信号在 80386 内部经二分频后, 成为 80386 芯片的内部时钟。对于 16MHz 的 80386, CLK2 为 32MHz。

(2) 数据总线 D<sub>31</sub>~D<sub>0</sub> (双向、三态)。80386 有 32 根数据总线, 数据总线宽度可以动态地在 16 位和 32 位之间进行切换。

(3) 地址总线 A<sub>31</sub>~A<sub>2</sub> (输出、三态)。表示地址的高 30 位。

(4) 字节选择  $\overline{BE}_3 \sim \overline{BE}_0$  (输出)。以字节为单位,  $\overline{BE}_0$  选择 D<sub>7</sub>~D<sub>0</sub>,  $\overline{BE}_1$  选择 D<sub>15</sub>~D<sub>8</sub>……如果传送地址 0 中的单字节数, 则  $\overline{BE}_0$  起作用; 如果从地址 0 读取双字 (4 个字节), 则  $\overline{BE}_0 \sim \overline{BE}_3$  都起作用, 这样一次可传送 32 位 (D<sub>31</sub>~D<sub>0</sub>) 数据。

(5) 总线状态输出 (三态)。这些三态输出信号用来定义总线周期的类型。W/ $\overline{R}$  用于区分写总线周期/读总线周期; D/ $\overline{C}$  用于区分数据总线周期/控制总线周期; M/ $\overline{IO}$  用于区分访存总线周期/访 I/O 总线周期;  $\overline{LOCK}$  用于区分锁定总线周期/开启总线周期;  $\overline{ADS}$  用来确定总线周期的开始, 即表示 A<sub>31</sub>~A<sub>2</sub>、 $\overline{BE}_3 \sim \overline{BE}_0$ 、W/ $\overline{R}$ 、M/ $\overline{IO}$ 、D/ $\overline{C}$  为有效。

(6) 总线控制输入。 $\overline{READY}$  为准备就绪信号, 是总线周期与外设同步的信号。NA 为下一地址请求信号, 表示可以接收下一个地址。BS16 为总线宽度信号, 要求数据总线为 6 位时使用。

(7) 协处理器接口信号 (输入)。PEREQ 为协处理器请求,  $\overline{BUSY}$  为协处理器忙,  $\overline{ERROR}$  为协处理器出错。

(8) 总线仲裁信号。总线请求 HOLD (输入) 和总线响应 HLDA (输出)。

(9) 中断输入信号。可屏蔽中断请求信号 INTR 和非屏蔽中断请求信号 NMI。

(10) 复位信号 (输入)。当 RESET 信号输入时, 将 80386 置成预定的复位状态。

### 2.6.3 80486 的体系结构

80486 是 Intel 公司于 1989 年推出的 32 位高档微处理器, 它在 80386 的基础上做了一些改进。简单地说, 80486 芯片相当于一块 80386 加上一块 80387 (数学协处理器), 再加上 8KB 的内片快速缓存 (Cache)。

80486 拥有 80386 的所有功能, 诸如页式存储管理、段式存储管理、DEBUG 功能、自测试功能、三种工作模式、多任务、流水线指令执行方式和 32 位整数算术逻辑运算等。

80486 完全和 80386 兼容, 目标码一级也兼容。在软件上, 80486 实际上和 80386 一样, 区别主要表现在底层硬件实现上的不同。80486 可以用于高档微机和工作站, 它的属性能使它在 DOS、OS/2、Windows 和 UNIX 系统上得到广泛的应用。

和 80386 相比, 80486 具有下述特点:

① 80486 在 Intel CPU 的历史上首次采用了 RISC 技术, 常用指令仅需一个时钟周期即可完成。RISC 即 Reduced Instruction Set Computer (精简指令集计算机), 与其相对的就是传统的复杂指令集计算机 CISC (Complex Instruction Set Computer)。80486 由于要与 80386 兼容, 所以指令集并未精简, 主要采用 RISC 技术。RISC 技术的特点是并行处理技术, 采用流水线结构、多级 Cache 存储器结构和对称多处理技术。80486 采用 RISC 技术的目的, 是要达到一个时钟周期执行一条指令。80486 CPU 一般都达到了这一设计目标, 平均一个时钟周期执行 1.2 条指令。

② 80486 采用突发总线 (Burst Bus) 同 RAM 进行高速数据交换。通常 CPU 同 RAM 交换数据时, 采用的是取得一个地址, 交换一个数据, 再取得一个地址, 再交换一个数据……而采用突发总线后, 每取得一个地址, 则这个地址及其后地址的数据都一起进行交换。这种技术特别适用于图形显示, 因为图形显示的地址空间一般都是连续的; 其次是网络应用, 因为网络上经常需要进行一连串地址空间的数据传送。突发总线的传送效率极大地提高了 80486 微处理器的性能。

③ 80486 CPU 将数学协处理器、Cache 及 Cache 控制器一起集成到片内, 极大地提高了 CPU 的处理速度。随着 CPU 工作时钟的提高, 一般动态 RAM 芯片的存取速度相对较慢, 不能跟上快速 CPU 的速度, 致使 CPU 出现等待状态。为此, 在 80386 系统中, 采用在主机板上设置高速缓冲存储器 (Cache RAM) 的方法, 使 CPU 在大多数情况下能够快速地访问最近使用过的指令和数据, 实现零等待, 从而大大地提高了系统的性能。

80486 把数学协处理器和 Cache 都集成在片内, 并且数学协处理器 387DX 与 RISC CPU 之间, 以及 Cache 与 RISC CPU 之间均用片内高速数据总线 (HS-BUS) 进行数据传送。387DX 与 RISC CPU 之间的高速总线是 64 位的, 而 Cache 同 RISC CPU 之间则是 128 位的。这种带宽的数据交换通道是板上协处理器和 Cache 所无法相比的。目前 386 主机板上的 Cache 与 CPU 的数据交换通道一般是 32 位的, 最高可以到 64 位。

由于以上特点, 在相同的时钟频率下, 80486 CPU 的处理速度一般比 80386 快 2~3 倍。即使是时钟频率为 25MHz 的 486SX-25, 在运行 Word Perfect 5.1、Page Maker 4.0、Microsoft 6.0 和 AUTOCAD 11.0 等 8 种典型应用程序时, 其平均运行速度是时钟频率为 40MHz 的 386DX-40 的 1.22 倍。

80486 微处理器有下列四种型号：

① 80486SX。这是 Intel 80486 CPU 的入门产品，片上不含数学协处理器。它兼有 486 的性能和 386 的价格两大优点。目前 80486SX 有 16MHz、20MHz、25MHz、33MHz 和 40MHz 等几种产品。

② 80486DX。它属于 486 家族中的中档产品，与 80486SX 相比增加了片上数学协处理器的浮点运算动能。80486DX 的时钟频率一般较高，有 25MHz、33MHz、40MHz 和 50MHz 等几种产品。

③ 80486DX2。它是 486 家族中的高档产品，它在 80486DX 的基础上采用了倍速技术（Speed Doubling），使 CPU 的执行速度双倍于系统总线的速度。目前 80486DX2 主要有 50MHz 和 66MHz 两种产品（相应的系统总线分别是 25MHz 和 33MHz）。

④ Over Drive。它是 Intel 提供的一种升级产品，为 80486SX、80486DX 和 80486DX2 提供升级服务，使之用比较少的花费就能得到更高的系统性能。

## 2.6.4 Pentium微处理器的体系结构

### 1. Pentium微处理器的特点

586 CPU 问世，并被命名为 Pentium（奔腾）。Pentium 最初级的 CPU 分别工作在与系统总线频率相同的 60MHz 和 66MHz 两种频率下。早期的奔腾 75~120MHz 使用 0.5mm 的制造工艺，后期 120MHz 以上的奔腾 CPU 则改用 0.35mm 工艺。

Pentium Pro 是第一代产品，cache 有 256KB 或 512KB，最大有 1MB。工作频率有：133/66MHz、150/60MHz、166/66MHz、180/60MHz、200/66MHz。

Pentium II 有几种不同核心结构的系列产品，其中，第一代采用 Klamath 核心，0.35mm 工艺制造，内部集成了 750 万个晶体管，核心工作电压为 2.8V。Pentium II 微处理器采用了双重独立总线结构，其中一条总线连通二级缓存，另一条主要负责内存。Pentium II 使用了一种脱离芯片的外部高速 L2 Cache，容量为 512KB，并以 CPU 主频的一半速度运行。L1 Cache 从 16KB 增至 32KB。

Pentium III 采用 0.25mm 工艺制造，内部集成了 950 万个晶体管；系统频率为 100MHz；采用第六代 CPU 核心 P6 微架构，针对 32 位应用程序进行优化，双重独立总线；L1 Cache 为 32KB，L2 Cache 大小为 512KB；采用 SECC2 封装形式；新增加了能够增强音频、视频和 3D 图形效果的数据流单指令多数据扩展 SSE 指令集，共 70 条新指令。

Pentium IV，实现了处理器性能和频率的大幅提高。微处理器从 Pentium Pro 到 PentiumII、PentiumIII 和 Celeron，英特尔只是通过改变微处理器的频率、二级缓存的大小、产品制造工艺来不断提高微处理器的性能，其内部结构并未改变。Pentium IV 则不同，它采用了全新的内核结构。处理器的算术逻辑单元运行在两倍的核心频率上，允许在 1/2 时钟周期里执行某一指令，大大降低了运算中的延迟；采用新的解码结构，可以更加有效地使用 Cache 的存储区；SSE2 指令集。SSE2 在 MMX 和 SSE 指令集的基础上新增 144 条指令集，提高了应用程序的处理能力。Pentium IV 处理器拥有 4200 万个晶体管，数据传输速率可以达到 44.8Gb/s，Pentium IV 处理器为因特网、图形处理、数据流视频、语音、3D 和多媒体等多种应用模式提供了强大功能。



## 2. 多核微处理器的特点

英特尔酷睿微体系结构是基于英特尔架构的台式、移动式和主流服务器多核处理器的新基础。多核是计算机体系结构发展史上划时代的革命性里程碑，多核处理器实现了真正的并行，极大地提升了 CPU 的性能。

14 级指令执行流水线的设计，大大降低了延迟；核心架构内建了 4 组指令编码器，以及 3 个算术逻辑单元，使得处理器的处理性能大大提高；共享二级缓存的设计，两个核心可根据需要直接调用缓存数据，提高了缓存容量的利用率，加快了处理速度。

处理器的微架构拥有宽位动态执行、智能功率能力、先进智能缓存、智能内存访问以及高级数字媒体增强五大创新功能，确保了处理器在拥有强劲性能的同时，还能够达到很高的能源效率。

## 3. 英特尔酷睿微体系结构的特点

在酷睿 2 处理器上，英特尔全面应用了 65 纳米的工艺制程，包含 2.91 亿个晶体管，采用了高性能低功耗晶体管等材料，其能效比奔腾处理器高出 40%，功耗降低 40%。通过共享二级缓存，降低功耗，进一步提升双核处理器的性能。

（1）数据处理能力。英特尔对 CPU 的性能提出了新标准，Core 微体系结构中的“宽位动态执行”拥有 4 组解码器，通过提升每个时钟周期完成的指令数，从而显著改进执行能力；增强的运算逻辑单元（ALU），以进一步支持微融合。它能够在单个周期内执行组合的指令对，从而使性能得到提升。

（2）功率能力。功率能力是一组旨在降低功耗和设计要求的特性。该特性可以管理所有处理器执行内核运行时的功耗。它含有一项高级功率门控能力，该能力可以在仅需要的单独处理器逻辑子系统上运行极其高效的逻辑控制。

（3）高速缓存。高速缓存是多核优化的高速缓存。英特尔在两个内核之间共享了二级高速缓存。通过在每个内核之间共享二级高速缓存，英特尔高速缓存还可以让每个内核动态地利用高达 100% 的可用二级高速缓存。当一个内核只需要较少的高速缓存时，其他内核便可以增加其占用二级高速缓存的百分比，以减少高速缓存错误并提高性能。多核优化的高速缓存还能够以更高的吞吐率获取高速缓存中的数据。

（4）内存访问。内存访问可以提高系统的性能，因为它能够优化内存子系统对可用数据带宽的使用，并隐藏内存访问的延迟。该目标是为了确保能够尽快地使用数据，并使该数据尽可能地用于需要的地方，以将延迟最小化，从而提高效率和速度。

（5）数字媒体增强。数字媒体增强是一项可以显著提高执行 SIMD 流指令扩展（SSE）指令性能的特性。128 位 SIMD 整数算法和 128 位 SIMD 双精度浮点操作减少了执行特定程序任务所需的全部指令数，将能够促使整体性能的增高。目前 SSE 指令集已普遍用于主流的软件中，包括绘图、影像、音频、加密、数学运算等，酷睿 2 处理器在家用多媒体和游戏的运行上将会广泛应用。

目前市场推出的 Intel 酷睿 2 双核 E8400，标称频率 3000MHz，二级缓存 6MB，前端总线 1333MHz，制程工艺 45 纳米；酷睿 i7-965Extreme，核心数四核，标称频率 3.2GHz，二级缓存 8MB，制程工艺 45 纳米，耗电 130W。

## 习题

1. 8086/8088 CPU 的地址线有多少位？其寻址范围是多少？
2. 8086/8088 CPU 分为哪两个部分？各部分的主要作用是什么？
3. 8086/8088 CPU 中指令队列有什么作用？其长度分别是多少？
4. 简述 8086/8088 CPU 的寄存器结构。
5. 简述 8086/8088 CPU 的标志寄存器中各位的作用。
6. 8086 与 8088 CPU 有何区别？
7. 对于 8086/8088 CPU，堆栈的位置如何确定？
8. 8086/8088 CPU 工作在最小模式：
  - (1) 当 CPU 访问存储器时，要利用哪些信号？
  - (2) 当 CPU 访问外设接口时，要利用哪些信号？
  - (3) 当 HOLD 有效并得到响应时，CPU 中的哪些信号置高阻抗状态？
9. 什么是逻辑地址？什么是物理地址？其关系如何？
10. 求出 1278H+3469H 的结果对标志寄存器各位的影响。
11. 简述 Pentium 微机的特点。

# 第 3 章

## 指令系统及汇编语言程序设计

### 教学目的和要求

不同种类微型计算机的指令系统是不同的，本章将详细介绍 8088/8086 指令系统的寻址方式、各种指令格式及功能、伪指令、汇编语言程序设计。要求读者熟悉指令，掌握指令寻址方式，熟悉汇编语言程序设计的基本方法，能编写出简单的汇编语言程序。

### 3.1 寻址方式

一条指令主要有两部分组成，一部分是指令的操作码，规定指令执行什么样的操作；另一部分是操作数，这部分规定了操作数本身或操作数的地址。如何寻找操作数就是寻址方式。

#### 1. 立即寻址

这种寻址方式所提供的操作数直接包含在指令中，这种寻址方式的操作数叫做立即数，可以是 8 位的，也可以是 16 位的。例如：

```
MOV AL, 20H      ; 将十六进制数 20H 送入 AL
MOV AX, 2845H     ; 将 2845H 送入 AX, AH 中为 28H, AL 中为 45H
```

#### 2. 直接寻址

数据在存储器中，存储单元的有效地址由指令直接指出，这种寻址方式叫做直接寻址。例如：

```
MOV AX, [1050H]   ; 将 DS 段的 1050H 和 1051H 两单元的内容送入 AX
```

比如，上一条指令执行时，设 DS=2000H，则执行过程是将绝对地址为 21050H 和 21051H 单元的内容送入 AX。

#### 3. 寄存器寻址

操作数在 CPU 的内部寄存器中，寄存器名在指令中指出，这种寻址方式就叫做寄存器寻址。对 8 位操作数来说，寄存器可以为 AH, AL, BH, BL, CH, CL, DH, DL。对 16 位操作数来说，寄存器可以为 AX, BX, CX, DX, SI, DI, SP 或 BP。例如：

```
MOV DH, CL        ; 将 CL 的内容送入 DH
MOV AX, BX         ; 将 BX 的内容送入 AX
INC CX             ; 将 CX 的内容加 1
```

#### 4. 寄存器间接寻址

采用寄存器间接寻址方式时，操作数一定在存储器中，存储单元的有效地址由寄存器指出，这些寄存器可以为 BX, BP, SI 和 DI 之一，即有效地址等于其中一个寄存器的值：

$$EA = \begin{cases} [BX] \\ [BP] \\ [SI] \\ [DI] \end{cases}$$

如果指令前面没有用前缀指明具体的段寄存器，则寻址时默认的段寄存器通常为 DS。如寄存器为 BP 时，则对应的段寄存器为 SS。

### （1）以BX寄存器进行间接寻址

例如： `MOV AX, [BX]`

设 `DS=2000H`, `BX=4000H`, 则本指令在执行时, 将 `24000H` 和 `24001H` 两单元的内容送入 `AX`。如果要对其他段寄存器所指的区域进行寻址, 则必须在指令前用前缀指出段寄存器名。

例如： `ES: MOV CX, [BX]`

设 `ES=2000H`, `BX=3000H`, 则本指令在执行时, 将 `23000H` 和 `23001H` 两单元的内容送入 `CX`。

### （2）以BP寄存器进行间接寻址

例如： `MOV BX, [BP]`

设 `SS=3000H`, `BP=5000H`, 则本指令在执行时, 将 `35000H` 和 `35001H` 两单元的内容送入 `BX`。

## 5. 变址寻址

上述的可以作为寄存器间接寻址的 4 个寄存器 `SI`, `DI`, `BX`, `BP`, 也可以做变址寻址。所谓变址寻址就是以指定的寄存器内容, 加上指令中给定的 8 位或 16 位位移量（当然要以一段寄存器作为地址基准），作为操作数的地址。

例如： `MOV AX, COUNT[SI]`

设 `DS=3000H`, `SI=2000H`, `COUNT=3000H`, 则本指令在执行时, 将 `35000H` 和 `35001H` 两单元的内容送入 `AX`。

## 6. 基址加变址的寻址

将 `BX`, `BP`, `SI` 和 `DI` 寄存器组合起来进行间接寻址——基址加变址的寻址。通常把 `BX` 和 `BP` 看做基址寄存器, 把 `SI`, `DI` 看做变址寄存器。这种寻址方式是把一个基址寄存器（`BX` 或 `BP`）的内容加上一个变址寄存器（`SI` 和 `DI`）的内容, 或再加上指令中指定的 8 位或 16 位位移量（当然要以一段寄存器作为地址基准），作为操作数的地址。

例如： `MOV AX, [BX+SI]`

设 `DS=3000H`, `BX=5000H`, `SI=2000H`, 则上面指令在执行时, 将 `37000H` 和 `37001H` 两单元的内容送入 `AX`。

例如： `MOV AX, [BX+SI+MASK]`

设 `DS=3000H`, `SI=1000H`, `BX=2000H`, `MASK=0250H`, 则上面指令在执行时, 将 `33250H` 和 `33251H` 两单元的内容送入 `AX`。

在通常情况下, 由基址地址决定哪一个段寄存器作为地址指针。若用 `BX` 作为基址地址, 则操作数在数据段区域中; 若用 `BP` 作为基址地址, 则操作数在堆栈段区域中。但若在指令中规定段是超越的, 则可用其他段寄存器作为地址基准。

## 3.2 指令系统

为了熟悉指令, 学会编写程序, 本节对 8088（8086）指令系统进行简要介绍。8088（8086）的指令系统可以分成以下 7 类：数据传送指令, 算术运算指令, 逻辑运算和移位指令, 串操作指令, 输入/输出指令, 控制转移指令, 处理器控制指令。

### 3.2.1 数据传送指令

#### 1. MOV OPRD1, OPRD2

MOV 是操作码，OPRD1 和 OPRD2 分别是目的操作数和源操作数。该指令把一个字节或一个字操作数从源地址传送到目的地址。

源操作数可以是累加器、寄存器、存储器以及立即操作数，目的操作数可以是累加器、寄存器和存储器。

数据传送指令列举如下：

(1) CPU 内部寄存器之间的数据传送（除段寄存器 CS 和指令指针 IP 以外）。例如：

```
MOV AL, BL
MOV DH, AL
MOV AX, BX
MOV DS, AX
MOV BX, DI
MOV DX, ES
MOV SI, BP
```

(2) 立即数传送至 CPU 内部的通用寄存器组（即 AX, BX, CX, DX, BP, SP, SI, DI），给寄存器赋值。例如：

```
MOV BL, 04H
MOV AX, 2400H
MOV SI, 3000H
```

(3) CPU 内部寄存器（除了 CS 和 IP 以外）与存储器之间的数据传送。可以实现一个字节或一个字的传送。

- CPU 的通用寄存器与存储器之间的数据传送。例如：

```
MOV AL, BUFR
MOV AX, [SI]
MOV [DI], CX
MOV SI, BLOCK[BP]
```

- 段寄存器（除了 CS 以外）与存储器之间的数据传送。例如：

```
MOV DS, DATA[SI+BX]
MOV DEST[BP+DI], ES
```

需要注意的是，MOV 指令不能实现存储单元之间的数据传送，但我们可以用 CPU 的内部寄存器为桥梁来解决这一问题。例如：

```
MOV AL, AREA1
MOV AREA2, AL
```

程序中的增量 (INC)、减量 (DEC) 指令及转移指令 (JNZ)，我们将在后面介绍。其中 **OFFSET AREA1** 是指地址单元 **AREA1** 在段内的地址偏移量。找内存操作数时，必须以段地址加上此单元的段内地址偏移量，才能确定某一内存单元的物理地址。

堆栈是内存的一个区域，8088 中规定堆栈设置在堆栈段（SS）内，堆栈指针 SP 始终指向堆栈的顶部，由 SS 的内容和 SP 的内容确定堆栈区中的某一地址单元。堆栈的操作具有“后进先出”的特点。

堆栈操作指令中的操作数可以是段寄存器（除 CS）的内容、16 位的通用寄存器，以及内存的 16 位字。

当执行完以上两条压入堆栈指令时，堆栈中的内容如图 3-1 所示。

执行以上两条压入堆栈指令的工作过程:

图 3-1 堆栈操作示意图

⑦  $SP-1 \rightarrow SP$  ; 栈顶

⑧  $BL \rightarrow (SP)$

弹出堆栈的过程与此相反。例如：

**POP BX**

弹出的工作过程：

①  $(SP) \rightarrow BL$

②  $SP+1 \rightarrow SP$

③  $(SP) \rightarrow BH$

④  $SP+1 \rightarrow SP$

由此可见，SP 的内容始终指向堆栈的顶。

### 3. 交换指令

**XCHG OPRD1 (目的), OPRD2 (源)**

交换指令，可以把一个字节或一个字的源操作数与目的操作数交换。这种交换能在通用寄存器与累加器之间、通用寄存器之间、通用寄存器与存储器之间进行，但段寄存器不能作为一个操作数。例如：

**XCHG AL, DL**

**XCHG BX, SI**

**XCHG AX, DX**

**XCHG AX, DATA[SI]**

### 4. 换码指令XLAT

换码指令是完成一个字节的查表转换指令。累加器 AL 的内容作为一个表的下标，这个表的基地址在 BX 寄存器中，转换后的一个字节的操作数放在 AL 中。例如：

**MOV BX, TABLE** ; 将表格首地址送 BX

**MOV AL, DATA** ; 序号送 AL

**XLAT** ; 将 DS: (BX+AL) 对应的存储单元内容送 AL

### 5. 目标地址传送指令

目标地址传送指令是专用于传送地址码的指令，可用来传送操作数的段地址或偏移地址，共有 3 条指令。

#### (1) LEA 指令

**LEA OPRD1, OPRD2**

该指令把源操作数 OPRD2 的地址偏移量传送至目的操作数 OPRD1，源操作数必须是内存单元地址，目的操作数必须为一个 16 位的通用寄存器。这条指令常用来使一个寄存器作为地址指针。例如：

**LEA AX, [2000H]** ; 把地址偏移量 2000H 送入 AX 中



### （2）LDS指令

该指令完成一个地址指针的传送。地址指针包括段地址和地址偏移量。目的段地址送入 DS，地址偏移量送入一个 16 位的指针寄存器或变址寄存器。例如：

```
LDS    DI, [2800H]
```

执行上面的指令，将 2800H 和 2801H 地址单元中的内容（地址偏移量）送入 DI，将 2802H 和 2803H 地址单元中的内容（段地址）送入 DS。

### （3）LES指令

LES 指令除了把目标段地址送入段寄存器 ES 以外，其他与 LDS 指令类似。

## 6. 标志寄存器传送指令

### （1）LAHF

执行 LAHF 指令，将标志寄存器中的低 8 位传送到 AH 中。

### （2）SAHF

执行 SAHF 指令，将 AH 的内容传送到标志寄存器的低 8 位。

### （3）PUSHF

执行 PUSHF 指令，将标志寄存器中的内容压入堆栈，堆栈指针 SP 的值减 2。

### （4）POPF

执行 POPF 指令，将堆栈弹出的一个字放入标志寄存器，堆栈指针 SP 的值加 2。

## 3.2.2 算术运算指令

8088 提供加、减、乘、除 4 种基本算术运算指令。这些操作指令都可用于字节或字运算，也可以用于带符号数与无符号数的运算。8088 提供了各种校正操作，可以进行十进制的算术运算。

### 1. 加法指令

#### （1）ADD OPRD1, OPRD2

这是一条不带进位的加法指令，完成两个操作数的相加，结果送至目的操作数 OPRD1，即  $OPRD1 \leftarrow OPRD1 + OPRD2$ 。目的操作数可以是累加器、任一通用寄存器或存储器中的操作数。例如：

```
ADD    AL, 20H
ADD    AX, 2400H
ADD    DI, SI
ADD    [BX+DI], AX
ADD    AX, DATA[BX+SI]
```

#### （2）ADC指令

带进位位的加法指令，ADC 和 ADD 指令功能上类似，但 ADC 指令执行时，进位标志 CF 的值参加运算。即  $OPRD1 \leftarrow OPRD1 + OPRD2 + CF$ 。例如：

```
ADC    AL, 08H
```

```

ADC  AX, [SI]
ADC  BX, 2000H
ADC  DX, BX
ADC  AL, [BX]

```

【例 3-2】两个四字节的无符号数相加，这两个数分别放在 2000H 和 4000H 开始的存储单元中，低位在前，高位在后，要求相加之后所得的和放在 2000H 开始的内存单元中。

程序如下：

CLC		；清进位位 CF
MOV	SI, 2000H	；第一个数的首地址
MOV	AX, [SI]	；第一个数的低 16 位送入 AX
MOV	DI, 4000H	；第二个数的首地址
ADD	AX, [DI]	；第一个数或第二个数的低 16 位相加
MOV	[SI], AX	；相加后的和送入 2000H 和 2001H 单元
MOV	AX, [SI+2]	；第一个数的高 16 位送入 AX
ADC	AX, [DI+2]	；两个数的高 16 位连同 CF 位相加
MOV	[SI+2], AX	；高 16 位相加后的和送入 2002H 和 2003H 单元

### (3) 增量指令 INC

增量指令 INC 只有一个操作数，指令在执行时，将操作数的内容加 1，再送回该操作数。这条指令一般用在循环程序中修改指针和循环次数。例如：

```

INC  AL
INC  BX
INC  [SI]

```

INC 指令影响标志位 AF, OF, PF, SF, ZF, 但对 CF 无影响。

## 2. 减法指令

### (1) SUB OPRD1, OPRD2

SUB 为不带借位的减法指令，完成字节或字的相减： $\text{OPRD1} \leftarrow \text{OPRD1} - \text{OPRD2}$ 。

例如：

```

SUB  BX, CX
SUB  AL, 20
SUB  SI, 2000H
SUB  WORD PTR [DI], 2500H
SUB  [BP+2], CL

```

### (2) 带借位的减法指令 SBB

SBB 指令和 SUB 指令在功能上类似，只是 SBB 指令在进行减法运算时，还要减去 CF 的值。即： $\text{OPRD1} \leftarrow \text{OPRD1} - \text{OPRD2} - \text{CF}$ 。例如：

```
SBB  AX, 2000H
SBB  WORD PTR [DI+2], 3000H
```

### （3）减量指令DEC

减量指令只有一个操作数，执行时，将操作数的值减1，再将结果送回。例如：

```
DEC  AX
DEC  CL
DEC  BYTE PTR [DI+2]
```

### （4）求补指令NEG

求补指令 NEG 对指令中给出的操作数取补码，再将结果送回。因为对一个操作数取补相当于用0减去此操作数，所以 NEG 指令执行的也是减法操作。例如：

```
NEG  AL
NEG  CX
```

### （5）比较指令CMP OPRD1, OPRD2

比较指令 CMP 也是执行两个数的相减操作，即：OPRD1-OPRD2。但不送回相减的结果，只是使结果影响标志位。例如：

```
CMP  AL, 38H
CMP  AX, SI
CMP  DX, DI
CMP  AX, DATA[BX]
CMP  AX, 2000H
```

一般情况下，CMP 指令后面经常会有一条条件转移指令，用来检查标志位的状态是否满足某种关系。CMP 指令的执行会影响标志位 AF, CF, OF, PF, SF, ZF。

当两个带符号数 A 和 B 相比较时，只有在  $OF \oplus SF = 0$  时，才有  $A > B$ 。

## 3. 乘法指令

8088 在执行乘法指令时，有一个乘数总是放在 AL（8 位）或 AX 中。另外，将 DX 寄存器看成是 AX 寄存器的扩展，因此，当得到 16 位的乘积时，结果就在 AX 中。而得到 32 为乘积时，结果在 DX 和 AX 两个寄存器中，DX 中为乘积的高 16 位，AX 中为乘积的低 16 位。

#### （1）无符号数乘法指令MUL

① 8 位乘法：被乘数隐含在 AL 中，乘数为 8 位寄存器或存储单元的内容，乘积一定在 AX 中。例如：

```
MUL  BL
```

② 16 位乘法：被乘数隐含在 AX 中，乘数为 16 位寄存器或两个存储单元的内容，乘积的高 16 位放在 DX 中，低 16 位放在 AX 中。例如：

MUL CX

## (2) 带符号数乘法指令IMUL

带符号数的乘法指令为 IMUL，它在功能和形式上与 MUL 很类似，只是要求两个乘数均为有符号数。

## 4. 除法指令

### (1) 无符号数除法指令DIV

① 8 位除法：被除数隐含在 AX 中，除数可以是 8 位寄存器或存储单元的 8 位无符号数。指令执行结果：商在 AL 中，余数在 AH 中。例如：

```
MOV  AX, 1000
MOV  BL, 190
DIV  BL
```

则在 AL 中的商为 5，而 AH 中有余数 50。

② 16 位除法：被除数放在 DX、AX 连在一起的 32 位寄存器中（隐含）；除数为 16 位寄存器的内容或两个存储单元的内容构成的 16 位字；商在 AX 中，余数在 DX 中。例如：

```
MOV  AX, 1000
CWD
MOV  BX, 300
DIV  BX
```

执行结果为：AX=3，DX=100。

### (2) 带符号数除法指令IDIV

IDIV 在功能上和 DIV 很类似，只是把被除数和除数都看成有符号数。例如：

```
IDIV  BX
```

## 5. 校正指令

校正指令主要用于十进制数的调整。

① DAA 指令：对 AL 中的两个压缩十进制数的相加之和进行调整，得到压缩十进制和。

**【例 3-3】** 用 BCD 码实现 35H+48H 之和，结果送入 AL。  
程序如下：

```
MOV  AL, 35H
ADD  AL, 48H          ; AL=7DH
DAA                      ; 低四位加 6 修正, AL=83H
```

② DAS：对 AL 中的两个压缩十进制数相减之差进行调整，得到压缩十进制数。

③ AAA：对 AL 中 ASCII 码未压缩的十进制和进行调整。

④ AAS：对 AL 中 ASCII 码未压缩的十进制差进行调整。

- ⑤ AAD: 在除法指令前对 AX 中 ASCII 码未压缩的十进制数进行调整。
- ⑥ AAM: 对 AX 中的两个 ASCII 码未压缩十进制数的相乘结果进行调整。

### 3.2.3 逻辑运算和移位指令

#### 1. 逻辑运算指令

逻辑运算指令包括 AND（与）、OR（或）、NOT（非）、XOR（异或）和 TEST（测试）指令。AND、OR 和 XOR 指令的使用形式很相似，它们都是双操作数指令，既可对 8 位数操作，也可对 16 位数操作。

##### （1）AND指令

AND 指令对两个操作数进行按位的逻辑“与”运算，即只有相与的两位全为 1 时，与的结果才为 1，否则与的结果为 0。与的结果送至目的操作数。AND 指令的一般格式为：

AND     OPRD1, OPRD2

其中目的操作数 OPRD1 可以是累加器、任一通用寄存器、内存操作数。源操作数 OPRD2 可以是立即数、寄存器、内存操作数。例如：

```
AND    AL, 1FH
AND    AX, 00FFH
AND    AX, BX
AND    DX, [BX+SI]
AND    AL, BL
```

##### （2）OR指令

OR 指令对指定的两个操作数进行逻辑“或”运算。即进行或运算的两位中的任意一位为 1（或两位都为 1）时，或的结果为 1，否则为 0。或运算的结果送回目的操作数。OR 指令的一般格式为：

OR     OPRD1, OPRD2

其中，目的操作数 OPRD1 可以是累加器、通用寄存器、内存操作数。源操作数 OPRD2 可以是立即数、寄存器、内存操作数。例如：

```
OR     AL, 45H
OR     AX, 2435H
OR     BX, SI
OR     BUFFER[BX], SI
OR     AL, BL
```

##### （3）XOR指令

XOR 指令对两个指定的操作数按位进行“异或”运算。即进行异或运算的两位不相同（即一个为 1，另一个为 0），异或的结果为 1，否则为 0。异或运算的结果送回目的操作数。其一般格式为：

```
XOR    OPRD1, OPRD2
```

其中，目的操作数 OPRD1 可以是累加器、通用寄存器、内存操作数。源操作数可以是立即数、寄存器、内存操作数。例如：

```
XOR    AL, 1FH
XOR    AX, DX
XOR    BUFFER[BX], DI
XOR    BUFFER[BX+SI], AX
XOR    CX, 2000H
XOR    AX, AX
```

#### (4) NOT指令

NOT 指令对操作数进行按位求反操作，然后将结果送回。操作数可以是寄存器或存储器的内容。例如：

```
NOT    AL
NOT    BX
NOT    WORD PTR[2000H]
```

#### (5) TEST指令

TEST 指令的操作功能与 AND 指令相同，操作结果不送回，但结果将反映在标志位上。其一般格式为：

```
TEST    OPRD, im
```

其中，im 表示立即数，立即数中某一位为“1”，表示要对该位进行测试。例如：

```
TEST    AX, 8000H
```

如果 AX 内容中最高位为“1”，则标志位 ZF=0，否则 ZF=1。

例如：检测 AX 中的最高位是否为“1”，如果是“1”则转移。在这种情况下可以用以下指令：

```
TEST    AX, 8000H
JNZ     LOOP
.....
LOOP:   MOV    SI, 2000H
```

## 2. 移位指令

### (1) 非循环移位指令

8088 有 3 条移位指令：

- 算术左移和逻辑左移指令： SAL/SHL OPRD, m
- 算术右移指令： SAR OPRD, m
- 逻辑右移指令： SHR OPRD, m

上述指令中 OPRD 可以对寄存器操作数和内存操作数进行指定次数的移位，可以进行字节或字操作。指令格式中 m 表示移位次数，可以是 1 或寄存器 CL 中的内容。所有的移位指令在执行时都会影响标志位 CF，OF，PF，SF，ZF。

### ① 算术左移 SAL 和逻辑左移 SHL 指令

这两条指令的操作结果是完全一样的。每移位一次在右边最低位补一个 0，而左边的最高位则移入标志位 CF 中。如图 3-2 所示。

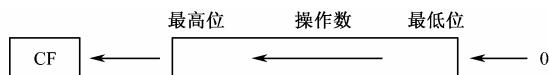


图 3-2 SAL/SHL 操作示意图

例如：SAL AL, 1  
SAL AX, CL  
SAL AL, CL

【例 3-4】 下面的程序段将 AL 中的数乘以 10。

```
SAL    AL, 1
MOV    BL, AL
MOV    CL, 2
SAL    AL, CL
ADD    AL, BL
```

### ② 算术右移指令 SAR

SAR 指令每执行一次移位操作，使操作数右移一位，但符号位保持不变，将最低位移到标志位 CF 中，如图 3-3 所示。

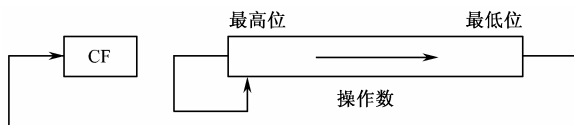


图 3-3 SAR 操作示意图

### ③ 逻辑右移指令 SHR

SHR 指令每执行一次移位操作，使操作数右移一位，将最低位移到标志位 CF 中，左边的最高位补 0，如图 3-4 所示。

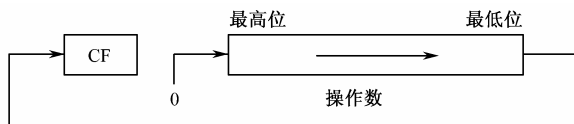


图 3-4 SHR 操作示意图

### (2) 循环移位指令

8088 有 4 条循环移位指令：

- 左循环移位指令:            ROL     OPRD, m
- 右循环移位指令:            ROR     OPRD, m
- 带进位左循环移位指令:    RCL     OPRD, m
- 带进位右循环移位指令:    RCR     OPRD, m

上述循环指令 OPRD 可以对寄存器操作数和内存操作数进行指定次数的移位。可以循环一次，也可以由寄存器 CL 的内容来决定循环次数。

循环指令可以进行字节操作，也可以进行字操作。循环移位指令操作，只影响标志位 CF 和溢出标志位 OF。

#### ① ROL 指令

执行 ROL 指令，每进行一次左移位，就将操作数的最高位移入 CF 中，并且将最高位移入操作数的最低位，从而构成一个左移小循环，如图 3-5 (a) 所示。当规定的循环次数为 1 时，若循环以后的操作数的最高位不等于标志位 CF，则溢出标志位 OF=1，否则 OF=0。这可以用来判断移位前后的符号位是否改变了。

#### ② ROR 指令

执行 ROR 指令，每进行一次右移位，就将操作数的最低位移入标志位 CF 中，同时将最低位移入操作数的最高位，从而构成了一个右移小循环，如图 3-5 (b) 所示。

#### ③ RCL 指令

执行 RCL 指令，每左移一次，就将操作数的最高位移入标志位 CF 中，而原来 CF 的内容则移入操作数的最低位，从而构成一个左移大循环，如图 5-5 (c) 所示。

#### ④ RCR 指令

执行 RCR 指令，每右移一次，就将标志位 CF 中的内容移入操作数的最高位，而操作数的最低位则移入标志位 CF 中，如图 3-5 (d) 所示。

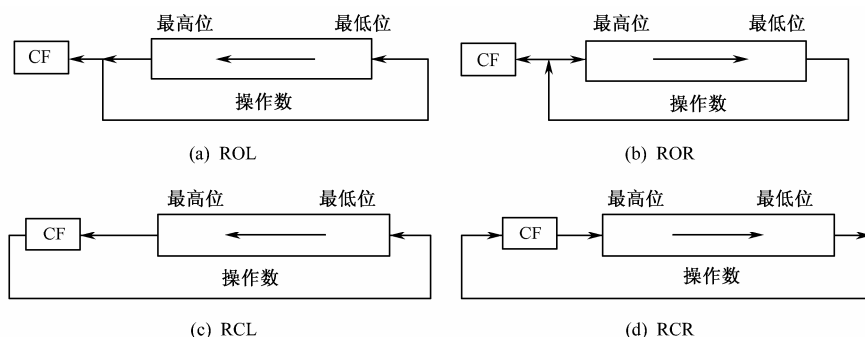


图 3-5 循环移位指令

### 3.2.4 串操作指令

串操作指令就是用一条指令实现对一串字符或数据的操作。串操作指令的特点是：

- ① 通过加重复前缀来实现串操作。
- ② 可以对字节串或字串进行操作。



③ 串操作指令都是用寄存器 SI 对源串寻址的，并且假定在 DS 段中；寄存器 DI 为目的的串寻址，并且假定在 ES 段中。串操作指令是一组源操作数和目的操作数都在存储单元中的指令。

④ 串操作时，地址的修改与方向标志 DF 有关。当 DF=1 时，SI 和 DI 做自动减量修改；当 DF=0 时，SI 和 DI 做自动增量修改。

### 1. MOVSB/MOVSW

字符串传送指令 MOVSB/MOVSW 将位于 DS 段、由 SI 所指定的存储单元的字节或字传送到位于 ES 段、由 DI 所指定的存储单元中，再修改 SI 和 DI，从而指向下一个元素。MOVSB 用于字节传送，MOVSW 用于字传送。MOVSB 或 MOVSW 指令前面通常加重复前缀 REP。例如：

MOV	SI, 2000H	；源地址为 2000H
MOV	DI, 3000H	；目的地址为 3000H
MOV	CX, 100	；字符串长度为 100
CLD		；DF=0，地址指针增量修改
REP	MOVSB	；将源地址开始的 100 个字节传送到目的地址单元中

由以上程序段可以看出，源地址由 SI 寄存器指定，目的地址由 DI 寄存器指定，并且默认源地址在 DS 段，目的地址在 ES 段；CX 寄存器中事先存放好要传送的字节数；使 CLD 指令将方向标志 DF 清 0。用 MOVSB 指令时，每传送一次，地址指针 SI 和 DI 自动加 1；用 MOVSW 指令时，每传送一次，地址指针 SI 和 DI 自动加 2 修改；对带 REP 重复前缀的串传送指令系统来说，每传送一次，CX 中的计数值总是减 1，与 DF 无关。

### 2. STOSB/STOSW、LODSB/LODSW

#### (1) STOSB/STOSW

该类指令是字符串存储指令。它将 AL 或 AX 中的数存在 ES 段 DI 寄存器所指的内存单元中，并且自动修改地址指针。加上前缀 REP 以后，用 STOSB 或 STOSW 指令可以使内存单元中填满相同的数。

【例 3-5】下面的程序段使 2000H 开始的 256 个单元清 0。

CLD		；清除方向标志
LEA	DI, [2000H]	；目的地址 2000H 送 DI
MOV	CX, 0080H	；共有 128 个字
XOR	AX, AX	；AX 清 0
REP	STOSW	；将 256 个字节清 0

#### (2) LODSB/LODSW

该类指令为取字符串指令。将位于 DS 段由 SI 所指定的存储单元中的内容取到 AL 或 AX 中。

### 3. CMPSB/CMPSW

字符串比较指令 CMPSB/CMPSW 把 DS 段由 SI 所指定的字节或字，与 ES 段由 DI 所

指定的字节或字相比较，并且在比较之后自动修改地址指针。通过重复前缀的控制，利用 **CMPSB** 或 **CMPSW**，可以实现在两个字符串中寻找第一个不相等的元素，或者第一个相等的元素。

【例 3-6】 对 **STRING1** 和 **STRING2** 两个字符串进行比较的程序。

```

MOV     SI, OFFSET STRING1
MOV     DI, OFFSET STRING2
MOV     CX, COUNT
CLD
REPZ    CMPSB      ; 当串未结束 (CX≠0) 且串相等 (Z=1) 继续比较
JNZ     UNMAT      ; 若串不相同，在 RESULT 单元中置 0FFH
MOV     AL, 0       ; 若串相等，在 RESULT 单元中置 “0”
JMP     OUTPT
UNMAT:  MOV     AL, 0FFH
OUTPT:  MOV     RESULT, AL
HLT

```

#### 4. SCASB/SCASW

字符串检索指令 **SCASB/SCASW** 用 **AL** 中的字节或 **AX** 中的字，与位于 **ES** 段由 **DI** 寄存器所指定的内存单元或字相比较。通过前缀的控制，可以实现在 **DI** 所指定的字符串中，寻找第一个与 **AL**（或 **AX**）的内容不同的字节（或字），或者寻找第一个与 **AL**（或 **AX**）的内容相同的字节（或字）。

【例 3-7】 把要搜索的关键字放在 **AL** 中，用以搜索内存的某一数据块中有无此关键字。若有把搜索次数记录下来（若次数为 0，表示无要搜索的关键字），且记录下存放关键字的地址。

程序段如下：

```

MOV     DI, OFFSET BLOCK      ; 字符串首地址送到 DI
MOV     CX, COUNT             ; 字符长度
MOV     AL, CHAR              ; 关键字
REPNE   SCASB                 ; 当串未结束 (CX≠0) 且串元素不等于搜索
                               ; 值 (Z=0) 时继续搜索
JZ      FOUND                 ; 搜索到关键字转移
MOV     DI, 0
JMP     DONE
FOUND:  DEC     DI
MOV     POINTR, DI            ; 记录下存放关键字的地址
MOV     BX, OFFSET BLOCK
SUB     BX, DI
MOV     DI, BX                ; 记录搜索次数
DONE:   HLT

```

### 3.2.5 输入/输出指令

输入/输出指令可以分为两类：一类是直接寻址的输入/输出指令，输入/输出端口地址由指令给出，可寻址 PORT0~PORT255，共 256 个输入端口。另一类是间接的输入/输出指令，端口地址在寄存器 DX 中，允许寻址 64K 个输入/输出端口。

#### 1. 直接寻址的输入/输出指令

输入指令允许把一个字节或一个字由输入端口传送到 AL 或 AX 中。例如：

```
IN    AL, 60H
IN    AX, 80H
```

输出指令是把 AL 中的一个字节或 AX 中的一个字，传送到输出端口。例如：

```
OUT   40H, AL
OUT   70H, AX
```

#### 2. 寄存器间接寻址的输入/输出指令

例如：MOV DX, 07F2H

```
IN     AL, DX
```

表示从 DX（07F2H）所指的端口中读取一个字节输入到 AL 中。

例如：MOV DX, 07F2H

```
OUT    DX, AX
```

表示将 AL 中的内容输出到 DX（07F2H）所指的端口中，同时将 AH 中的内容输出到 DX+1（07F3H）所指的端口中。

### 3.2.6 控制转移指令

#### 1. 转移指令和调用指令的寻址

##### （1）段内直接转移方式

采用段内直接转移方式时，指令中给出一个相对位移量，这样，有效转移地址为 IP 的当前内容再加上一个 8 位或 16 位的位移量。在条件转移指令中，只能用 8 位的位移量。

##### （2）段内间接转移方式

采用段内间接转移方式时，有效地址在寄存器中，此种寻址方式只适用于无条件转移指令。

##### （3）段间直接转移方式

采用段间直接转移方式时，指令中要给出转移地址的段值和偏移量。产生转移时，将段值代入 CS 中，将偏移量代入 IP 中。用这种寻址方式，可以提供一种使程序从一个代码段转移到另一个代码段的方法。

##### （4）段间间接寻址方式

在段间间接寻址方式下产生转移时，IP 和 CS 的内容用内存中 2 个连续的字来替换。这里要指出的是，段间转移和段内间接转移必须是无条件转移指令。

## 2. 程序调用和返回指令

指令系统中提供了子程序段内直接调用指令，段内间接调用指令，段间直接调用指令和段间间接调用指令。例如：

CALL 2000H	；段内直接调用，调用地址在指令中给出
CALL AX	；段内间接调用，调用地址由 AX 给出
CALL 2000H: 3000H	；段间直接调用，调用段地址和偏移量都在指令中给出
CALL DWORD PTR[DI]	；段间间接调用，调用地址在 DI、DI+1、DI+2、DI+3 所指的内存单元中，前 2 个字节为偏移量，后 2 个字节为段地址

调用指令在执行时，会把下一条指令的地址推入堆栈，这个地址是返回地址。在段内调用的情况下，只是把返回地址的偏移量推入堆栈；在段间调用的情况下，则把返回地址的段地址和偏移量都推入堆栈。

和调用指令相对应的是返回指令。返回指令是子程序的最后一条指令，是用来返回高一层的程序的指令。

返回指令在执行时，会从堆栈顶部弹出返回地址，返回指令的类型要和调用指令的类型相对应。如果一个子程序是段内调用的，那么末尾用段内返回指令，这种情况下返回时，从栈顶弹出 2 个字节作为返回地址的偏移量；如果子程序是段间调用的，那么末尾用段间返回指令，这种情况下返回时，从栈顶弹出 4 个字节，分别作为返回地址的偏移量和段地址。

## 3. 无条件转移指令和条件转移指令

### (1) 无条件转移指令

无条件转移指令可以转移到内存中的任何程序段。和调用指令类似，无条件转移指令也有 4 种形式。例如：

JMP 2000	；段内直接转移，转移地址的偏移量由指令给出
JMP CX	；段内间接转移，转移地址的偏移量由 CX 给出
JMP 3000H: D200H	；段间直接转移，段地址和偏移量由指令给出
JMP DWORD PTR[SI]	；段间间接转移，段地址和偏移量放在 SI, SI+1, SI+2, SI+3 这 4 个单元中，前 2 个单元的内容作为偏移量，后 2 个单元的内容作为段地址

### (2) 条件转移指令

8088 中有不同的条件转移指令。它们根据标志寄存器中各标志位的状态，决定程序是否进行转移。条件转移指令的目的地址必须在现行的代码段（CS）内，并且以当前指针寄存器 IP 的内容为基准，转移范围应在+127~-128 之内。

条件转移指令的具体形式如下：

JE/JZ	；结果为 0，则转移
JNE/JNZ	；结果不为 0，则转移
JG/JNLE	；大于，即不小于且不等于，则转移
JNG/JLE	；不大于，即小于或等于，则转移

JL/JNGE	； 小于，即不大于且不等于，则转移
JNL/JGE	； 不小于，即大于或等于，则转移
JB/JNAE	； 低于，即不高于且不等于，则转移
JNB/JAE	； 不低于，即高于或者等于，则转移
JA/JNBE	； 高于，即不低于且不等于，则转移
JNA/JBE	； 不高于，即低于或者等于，则转移
JS	； 符号标志位 SF 为 1，则转移
JNS	； 符号标志位 SF 为 0，则转移
JO	； 溢出标志位 OF 为 1，则转移
JNO	； 溢出标志位 OF 为 0，则转移
JP	； 奇偶标志位 PF 为 1，则转移
JNP	； 奇偶标志位 PF 为 0，则转移
JCXZ	； 如 CX 中的值为 0，则转移

【例 3-8】 设 4000H 开始的区域中，存放 64H 个数据，要求找出其中最大的一个数，并存到 4000H 单元。

程序如下：

	MOV	BX, 4000H	； BX 指向 4000H 单元
	MOV	AL, [BX]	； 取第一个数
	MOV	CX, 64H	； CX 作为计数器
P1:	INC	BX	； BX 指针指向下一个地址
	CMP	AL, [BX]	； 和下一个数比较
	JAE	P2	； 如比下一个数大或相等，则转到 P2
	MOV	AL, [BX]	； 如下一个数大，则将下一个数取到 AL
P2:	DEC	CX	； CX 中计数值减 1
	JNZ	P1	； 如不为 0，则转到 P1
	MOV	BX, 4000H	； 比较完毕，BX 指向 4000H
	MOV	[BX], AL	； 最大数送入 4000H 单元中

#### 4. 循环控制指令

在设计循环程序时，可以用控制指令来控制循环是否继续。这些控制指令的目的地址在+127~-128 范围内。

##### （1）LOOP 指令

每执行一次 LOOP 指令，CX 内容减 1，CX≠0 循环，CX=0 退出循环。

例如，用下面两条指令可以构成简单延时子程序：

	MOV	CX, 0AF3H	； 设循环次数
TABC:	LOOP	TABC	； CX 减 1，如不为 0，则循环
	⋮	}	

## (2) LOOPZ/LOOPE

LOOPZ 和 LOOPE 是同一条指令的两个不同的助记符。执行指令时,使 CX 减 1,判断 CX≠0 且 ZF=1 循环;两者之一或同时不满足,则停止循环。

## (3) LOOPNZ/LOOPNE

LOOPNZ/LOOPNE 是和 LOOPZ/LOOPE 相对应的一条指令。LOOPNZ/LOOPNE 在执行时,使 CX 减 1,判断 CX≠0 且 ZF=0 循环;两者之一或同时不满足,则停止循环。

## 5. 中断指令及中断返回指令

在 8088 系统中,当执行到中断指令 INT 时,便中断当前程序的执行,转向由 256 个中断向量所提供的中断入口地址之一去执行。

执行中断指令时,首先把当前的标志寄存器的内容推入堆栈,堆栈指针 SP 减 2,然后清除中断允许标志 IF 和单步标志 TF。CPU 将主程序的下一条指令的地址即断点地址的段值和偏移量推入堆栈,同时堆栈指针 SP 减 4。保护好断点以后,就转向中断向量,即中断类型号乘以 4 就是中断向量的存放地址,中断向量就是中断处理程序的入口地址。中断向量的每 4 个单元对应于 1 个中断。

例如,INT 21 指令提供的中断类型号为 21,由  $21 \times 4 = 84$ ,得到中断向量的存放地址为 84H~87H,设这 4 个单元中存放的值为 00、20、00、30,则 CPU 转到 3000H:2000H 单元去执行中断处理程序。

各中断处理程序的功能各不相同,但是,中断处理程序的最后一条指令总是 IRET。执行 IRET 指令时,先从堆栈中弹出 4 个单元的内容送入 IP 和 CS,从而恢复断点地址,然后弹出标志寄存器的值。

## 3.2.7 处理器控制指令

### 1. 标志位操作指令

该类指令共有 7 条,分别对 CF 位、DF 位及 IF 位进行操作。

- ① CLC: 此指令清进位标志 CF=0。
- ② CMC: 此指令将进位标志 CF 求反。
- ③ STC: 此指令置进位标志 CF=1。
- ④ CLD: 此指令清方向标志 DF=0,串操作时,使地址增量。
- ⑤ STD: 此指令置方向标志 DF=1,串操作时,使地址减量。
- ⑥ STI: 允许中断标志,使 IF=1,则 CPU 可以响应 INTR 外部中断请求。
- ⑦ CLI: 清中断标志,使 IF=0, CPU 不响应 INTR 外部中断请求。

### 2. 处理器暂停 HLT 指令

执行 HLT 指令使 8088 进入暂停状态。只有下面三种情况之一发生时, CPU 才脱离暂停状态: ① RESET 线有复位信号; ② NMI 线上有请示信号; ③ 中断允许 (IF=1), 在 INTR 线上有中断请求。

HLT 指令常在程序中为了等待中断而使用。

### 3. 等待指令WAIT

执行该指令，使 8088 处于空操作状态，不断测试 8088 的 TEST 线。若该线电平为 1，则继续等待；若为 0，则退出等待状态。该指令主要用于 8088 与协处理器和外部设备之间的同步。

### 4. 处理器交权指令ESC

ESC 指令为交权指令，当处理器执行这条指令时把控制权交给协处理器。

### 5. 封锁总线指令LOCK

LOCK 指令是一个前缀，可放在任何一条指令前面。这条指令执行时，就封锁了总线的控制权，其他的处理器将得不到总线控制权，这一过程一直持续到指令执行完毕为止。它常用于多机系统。

## 3.3 系统功能调用

MS DOS 系统中设置了几十个内部子程序，熟悉这些子程序的调用类型和方法，可以方便用户进行汇编语言程序设计。常用的软中断指令有 8 条，系统规定它们的中断类型码为 20H~27H，它们各自的功能及入口/出口参数如表 3-1 所示。

表 3-1 常用软中断功能及参数

软中断指令	功 能	入 口 参 数	出 口 参 数
INT 20	程序正常退出		
INT 21	系统功能调用	AH=功能号，相应入口号数	相应出口号
INT 22	结束退出		
INT 23	CTRL-BREAK 处理		
INT 24	出错退出		
INT 25	读盘	AL=驱动器号 CX=读入扇区数 DX=起始逻辑扇区号 DS: BX 内存缓冲区地址	CF=0 成功，CF=1 出错
INT 26	写盘	AL=驱动器号 CX=写入扇区数 DX=起始逻辑扇区号 DS: BX 内存缓冲区地址	CF=0 成功，CF=1 出错
INT 27	驻留退出		

其中 INT 21 系统功能调用是一种特殊的中断调用。

在 DOS 功能调用中，不同功能调用是用功能号来区分的，按功能不同，应事先将功能号放入 AH 中，在其他寄存器中应放入指定的调用参数。随后执行一条 INT 21 指令即可实现对应的功能调用。

下面将介绍几个常用的功能调用。

### 1. 键盘输入单字符

这是 1 号系统功能调用，可用如下指令：

```
MOV    AH, 1          ; 置功能号
INT     21             ; 输入结果放在 AL 中，并显示在屏幕上
```

### 2. 返回操作系统

```
MOV    AH, 4CH        ; 置功能号
INT     21             ; 返回操作系统，屏幕显示系统提示符
```

### 3. 输出单字符

```
MOV    DL, "A"        ; 字符送入 DL
MOV    AH, 2          ; 置功能号
INT     21             ; 字符显示输出（或打印）
```

### 4. 输出字符串

```
MOV    DX, OFFSET BUF ; 存放字符串缓冲区中首地址
MOV    AH, 9          ; 置功能号
INT     21             ; 从屏幕显示字符串（或打印），字符串以“$”字符为结束标志
```

### 5. 无回显键盘输入单字符

这是 8 号系统功能调用，等待从键盘输入单字符，将其 ASCII 码置入 AL 寄存器中，但不送至屏幕显示。

```
MOV    AH, 8
INT     21H
```

### 6. 设置日期

这是 2BH 号系统功能调用，其功能是设置有效日期。

【例 3-9】 设置日期为 2003 年 12 月 26 日，年号以 BCD 码形式置入 CX 寄存器，月号置入 DH 寄存器，日号置入 DL 寄存器。

程序如下。

```
MOV    CX, 2003H
MOV    DH, 12H
MOV    DL, 26H
MOV    AH, 2BH
INT     21H
```

## 3.4 汇编语言程序设计

用指令的助记符、符号地址、标号、伪指令等符号书写程序的语言称为汇编语言。用



汇编语言编写的程序叫做源程序。把汇编语言源程序自动翻译成机器语言（目的程序）的过程叫做汇编程序。汇编过程如图 3-6 所示。

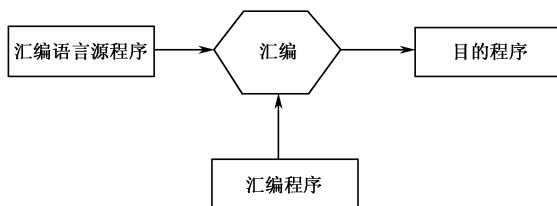


图 3-6 汇编过程

汇编语言和机器语言密切相关，它是面向机器的语言。CPU 不同的机器有不同的汇编语言。汇编语言的指令和机器语言的指令之间有一一对应的关系。采用汇编语言进行程序设计时，可以充分利用机器的硬件功能和结构特点，有效地加快程序的执行速度，减小目标程序所占的存储空间。和高级语言相比，汇编语言为程序员提供了直接控制目标代码的手段，而且可以对输入/输出端口进行控制，实时性能好，执行速度快，节省存储空间。所以，汇编语言大量用于编写计算机系统程序、实时控制程序、实时通信程序等。

### 3.4.1 汇编语言的语言格式

汇编语言编写的源程序是由许多语句组成的。每条语句由 1~4 个部分组成，格式如下：

[标号] 助记符 [操作数] [注释]

根据不同指令功能，其中用方括号括起来的部分，可以有，也可以没有。编写语句时每部分之间要用空格分开，这些部分可以在一行输入，以使用户阅读源程序。

#### （1）标号

这是给指令或某一存储单元地址所起的名称，常作为一段程序的开头，一个数据块的开头。

标号使用要注意：

- ① 标号可以由字母、数字和下划线组合，一个标号的最大长度不能超过 31 个字符。
- ② 标号不能以数字开头，但数字可以出现在标号符的中间或末尾。

#### （2）助记符

助记符是表示不同操作的指令，可以是指令的助记符，也可以是伪指令。

#### （3）操作数

操作数是指令执行的对象。

#### （4）注释

在汇编语言源程序中，为了便于理解和阅读程序，常常加上注释。注释要用分号“;”打头，在汇编过程中，对注释不做处理。

例如：

标号	助记符	操作数	注释
----	-----	-----	----

	MOV	AL, BL	; BL 中的 8 位数据送入 AL
	RET		; 子程序返回指令
STAT:	MOV	AX, DATA	; 以 STAT 标号为程序开头
ABC	EQU	2000H	; 将 2000H 赋值给 ABC

3.4.2 常数

汇编语言程序中经常使用的常数有以下几种：

- ① 二进制数：以字母 B 结尾的，由 0 和 1 组成的数字序列。如：10101011B。
- ② 八进制数：以字母 Q 或 O 结尾的 0~7 的数字序列。如：23Q, 24O。
- ③ 十进制数：0~9 的数字序列，可以用字母 D 结尾，也可以省略。如：183D。
- ④ 十六进制数：以 H 字母结尾的，由数字 0~9 和字母 A~F 组成的序列。如：

3ACFH, 0AFH。

凡是以字母 A~F 为起始的一个十六进制数，必须在其前面加数字“0”，否则汇编程序会认为是标识符。

- ⑤ 实数：它由整数、小数和指数 3 部分组成。如：5.421E-4
- ⑥ 字符串常数：用引号括起来的一个或多个字符，这些字符以 ASCII 码形式存储在内存中。如：“ABC”，在内存中就是 41H、42H、43H。

3.4.3 伪指令

伪指令用来为汇编程序提供某些信息，让汇编程序在汇编过程中执行某些特定的功能。如伪指令可以指定一个程序的数据段从哪里开始，可以指定堆栈区的大小，等等。伪指令与指令的本质差别是，在汇编过程中伪指令并不形成任何代码，不直接命令 CPU 去执行什么操作，从而方便用户进行汇编语言程序设计。

常用的伪指令有以下几种。

1. 标号赋值伪指令 EQU

EQU 伪指令用来对一个标号赋值。例如：

```
ABC      EQU      2400H
```

使 ABC 为数值 2400 H。

2. 定义存储单元的伪指令 DB、DW、DD、DQ、DT

伪指令 DB 和 DW 等用来给出程序中所需要的数据、字符串、地址表。该类伪指令用来为一个数据项分配存储单元，用一个符号名与这个存储单元相联系，且为这个数据提供一个任选的初始值。

DB 用来规定字节，DW 用来规定字，DD 用来规定双字，DQ 用来规定 4 个字，DT 用来规定 10 个字。例如：

```
DATA      DB      24H, 32H
```

表示从 DATA 地址单元开始，连续存入 24H, 32H，共占用 2 个字节的地址。

```
DATA    DW    4142H
```

汇编时会把 41H 与 42H 分别放到与 DATA 相联系的两个连续的字节单元中（一个字）。42H 放在地址低字节，41 放在地址高字节。

```
DATA    DB    ? , ?
```

汇编程序分配两个字节单元，以 DATA 地址单元开始的两个单元可预置任何内容。

```
STRING  DB    "ABCD"
```

以 STRING 为起始地址单元，为字符串中的每一个字符分配一个字节单元，字符串自左至右以字符的 ASCII 码按地址递增的排列顺序依次存放 41H, 42H, 43H, 44H。

```
BUFFER  DB    20H    DUP ( ? )
```

表示保留 20H 个字节，每个字节可预置任意内容。

```
BUFFER  DB    50      DUP ( 0 )
```

表示以 BUFFER 为首地址的 50 个字节中都存放同一数据 00H。

### 3. 定义存储单元类型的伪指令 BYTE、WORD、DWORD

利用这些伪指令，可以对存储单元的类型进行规定。例如：

```
MOV      BYTE    PTR[DI],00H
MOV      WORD    PTR[1000H],00H
JMP      DWORD   PTR[2000H]
```

第一个语句使 DI 所指的 1 个单元清 0；第二个语句使 1000H 所指的 1 个字即 2 个单元清 0；第三个语句使程序转移到另外一个段的某个单元，转移地址放在 2000H 开始的 4 个单元，前两个单元中的内容作为转移地址的偏移量，后面两个单元中的内容作为转移地址的段值。

### 4. 段定义伪指令 SEGMENT、ENDS、ASSUME、ORG

伪指令 SEGMENT 和 ENDS 总是成对使用的。用这一对伪指令可以将汇编语言源程序分成几个段，通常分为数据段、堆栈段和代码段。

【例 3-10】 数据段、堆栈段两段程序框架。

```
DATA    SEGMENT
M1     DW    2478H
M2     DW    3579H
P1     DW    ?
P2     DW    ?
DATA    ENDS
STACK   SEGMENT
        DW    20DUP ( ? )
```

```
STACK    ENDS
```

伪指令 **ASSUME** 用来告诉汇编程序，哪一个段为数据段，哪一个段为堆栈段，哪一个段为代码段。段寄存器 **CS**、**DS**、**SS**、**ES** 应具有符号段地址，但真正将段地址装入段寄存器还需要由传送指令在执行时赋值（**CS** 除外）。例如：

```
CODE     SEGMENT
        ASSUME     CS: CODE, DS: DATA, SS: STACK
        MOV        AX, DATA
        MOV        DS, AX
        MOV        AX, STACK
        MOV        SS, AX
        :
CODE     ENDS
```

**ORG** 伪指令用来规定目标程序存放单元的偏移量。例如，在源程序的第一条指令前用了如下伪指令：

```
ORG      2000H
```

汇编语言程序将把指令指针 **IP** 的值置成 **2000H**，即目标程序的第一个字节放在 **2000H** 处。

### 5. 定义过程的伪指令 **PROC**、**ENDP**、**NEAR**和**FAR**

伪指令 **PROC** 和 **ENDP** 总是成对出现的，这两条伪指令之间的内容就作为一个过程，即一个子程序。

伪指令 **NEAR** 或 **FAR**，指出一个子程序是段内调用子程序还是段间调用子程序。在主程序中遇到调用指令 **CALL** 时，如果对应的子程序头部标有 **FAR**，则产生一个段间调用地址，它包括 16 位的段地址和 16 位的偏移量；如果子程序头部标有 **NEAR**，则为段内调用。子程序过程的 **RET** 指令产生的代码为 **C3H**；子程序头部标有 **FAR**，则子程序的 **RET** 指令产生的代码为 **CBH**。

【例 3-11】 延时子程序。

```
SOFTDAY: PROC      NEAR
            MOV      BL, 64H
DELAY:     MOV      CX, 6451H
WAIT:      LOOP     WAIT
            DEC      BL
            JNZ      DELAY
            RET
SOFTDAY: ENDP
```

### 6. 程序结束伪指令 **END**

伪指令 **END** 是源程序的结束标志，该指令并不和其他伪指令成对使用。汇编程序在对

源程序进行汇编的过程中，如果遇到 **END**，便得知源程序到此结束。例如：

```
END    START
```

### 7. 宏指令伪指令

在汇编语言的源程序中，有的程序段要多次使用，为了简化程序书写，该程序段可以用一条宏命令来代替。汇编程序汇编到宏命令时，仍会产生源程序所需的代码。例如：

```
MOV     CL, 4
SAL     AL, CL
```

若这两条指令在程序中要多次使用，就可以用一条宏命令来代替。当然在使用宏命令前首先要对宏命令进行定义。例如：

```
SHIFT   MACRO
        MOV CL, 4
        SAL AL, CL
        ENDM
```

这样定义以后，凡是要使 **AL** 中内容左移 4 位的操作都可用一条宏命令 **SHIFT** 代替。

**MACRO** 是宏定义符，是和 **ENDM** 宏定义结束符成对出现的。这两者之间的部分就是宏体，也就是该宏命令要代替的那一段程序。

### 3.4.4 汇编语言源程序的结构

汇编语言源程序建立在段结构的基础上，一个段就是一些指令和数据的集合。所以一个汇编语言源程序，根据程序用途被划分成几段，如数据段、堆栈段、附加段和程序段（码段），用 **CS**、**DS**、**SS**、**ES** 段寄存器存放段值。这样就构造了源程序的基本格式：

```
DATA     SEGMENT
          :
          }          存放数据项的数据段
DATA     ENDS

EXTRA     SEGMENT
          :
          }          存放数据项的附加段
EXTRA     ENDS

STACK1    SEGMENT
          :
          }          设置堆栈段
STACK1    ENDS
```

```

CODE    SEGMENT
        ASSUME  CS: CODE   DS: DATA
        ASSUME  SS: STACK1 ES: EXTRA
BEING:   MOV  AX, DATA
        MOV  DS, AX
        :      }           存放指令序列
CODE    ENDS
        END    BEING

```

结合上面的源程序结构格式，需要说明如下：

- ① 互相配对的 **SEGMENT** 和 **ENDS** 前的标号必须一样。
- ② **ASSUME** 语言使汇编程序得知哪一段是数据段 (**DS**)，哪一段是堆栈段 (**SS**)，哪一段是附加段 (**ES**)，哪一段是代码段 (**CS**)。除 **CS** 段以外，各个段寄存器的实际值，还要用 **MOV** 指令来赋予。
- ③ **END BEING** 表示源程序结束。

### 3.4.5 汇编语言程序举例

【例 3-12】 实现  $C=A+B$ ，设  $A$ 、 $B$ 、 $C$  均为字变量。

```

DATA    SEGMENT
A        DW      2453H
B        DW      4336H
C        DW      ?
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS: CODE, DS: DATA
START:   MOV     AX, DATA      ; 置 DS 段初值
        MOV     DS, AX
        MOV     AX, A
        ADD     AX, B
        MOV     C, AX          ; C=A+B
        MOV     AH, 4CH        ; 返回 DOS
        INT     21
CODE    ENDS
        END     START

```

【例 3-13】 求 1~10 的累加，和送 **AL**。

```

CODE    SEGMENT
        ASSUME  CS: CODE

```

```
START:  MOV     CX, 10
        MOV     BL, 1
        XOR     AL, AL
AGAIN:  ADD     AL, BL
        INC     BL
        LOOP    AGAIN
        MOV     AH, 4CH
        INT     21H
CODE    ENDS
        END     START
```

【例 3-14】 实现两个 16 位二进制数的相乘。

```
DATA    SEGMENT
M1      DW      00FFH
M2      DW      00FFH
P1      DW      ?
P2      DW      ?
DATA    ENDS
STACK   SEGMENT
ST      DB      100 DUP ( ? )
TOP     EQU     LENGTH ST
STACK   ENDS
CODE    SEGMENT
        ASSUME  CS: CODE, DS: DATA, SS: STACE
STAT:   MOV     AX, DATA
        MOV     DS, AX
        MOV     AX, STACK
        MOV     SS, AX
        MOV     AX, TOP
        MOV     SP, AX
        MOV     BX, OFFSET M1
        MOV     AX, [BX]
        MOV     DX, 00
        MOV     BX, OFFSET M2
        MUL     [BX]
        MOV     BX, OFFSET P1
        MOV     [BX], AX
        MOV     BX, OFFSET P2
        MOV     [BX], DX
```

```

                HLT
CODE           ENDS
                END          STAT

```

【例 3-15】 二进制加法程序。两个多字节的二进制数分别在以 ADD1 和 ADD2 为首地址的存储单元中，两个数的字长度放在 CONT 单元，相加后的结果放在 SUM 为首地址的单元中。所有的数低字节在前，高字节在后。

程序如下：

```

                DATA        SEGMENT
ADD1            DB    FEH, 86H, 7CH, 44H, 56H, 1FH
ADD2            DB    56H, 49H, 4EH, 0FH, 9CH, 22H
SUM             DB    6 DUP (0)
CONT            DB    3
                DATA        ENDS
STACK           SEGMENT PARA STACK "STACK"
                DB          100 DUP (?)
STACK           ENDS
CODE            SEGMENT
                ASSUME CS: CODE, DS: DATA, ES: DATA, SS: STACK
MADDB:         MOV         AX, DATA
                MOV         DS, AX                ; 初始化数据段寄存器
                MOV         ES, AX                ; 初始化辅助段寄存器
                MOV         SI, OFFSET ADD1       ; 被加数地址→SI
                MOV         DI, OFFSET ADD2       ; 加数地址→DI
                MOV         BX, OFFSET SUM        ; 和地址→BX
                MOV         CL, BYTE PTR CONT    ; 字长度
                MOV         CH, 0
                CLC
MADDB1:        MOV         AX, [SI]
                ADC         AX, [DI]              ; 16 位相加
                INC         SI
                INC         SI
                INC         DI
                INC         DI
                MOV         [BX], AX              ; 相加结果送结果单元
                INC         BX
                INC         BX
                LOOP        MADDB1                ; 执行循环
                HLT

```



```
CODE      ENDS
          END      MADDB
```

【例 3-16】 多字节的 BCD 码相加的程序。

```
DATA      SEGMENT
FIRST      DB      11, 22, 33, 44          ; 第一个加数
SECOND     DB      55, 66, 77, 88          ; 第二个加数
SUM        DB      20 DUP ( ? )           ; 结果存放单元
DATA      ENDS
STACK     SEGMENT
STA       DB      20 DUP ( ? )             ; 设堆栈长度为 20 个字节
TOP       EQU      LENGTH STA
STACK     ENDS
CODE      SEGMENT
          ASSUME   CS: CODE, DS: DATA, SS: STACK
START:    MOV      AX, DATA                ; 设数据段寄存器的值
          MOV      DS, AX
          MOV      AX, STACK                ; 设堆栈段寄存器的值
          MOV      SS, AX
          MOV      AX, TOP                  ; 设堆栈指针
          MOV      SP, AX
          MOV      SI, OFFSET FIRST         ; SI 指向第一个加数
          MOV      DI, OFFSET SUM           ; DI 指向结果单元
          MOV      BX, OFFSET SECOND        ; BX 指向第二个加数
          MOV      CX, 04                   ; 长度为 4 个字节
          CLD                               ; 清方向标志
          CLC                               ; 清进位标志
ADIT1:    CALL     AAA                      ; 完成多字节加法
          LOOP     ADIT1
          ⋮                                ; 主程序后续部分
AAA       PROC     NEAR                    ; 单字节加法子程序
          LODSB                               ; 取第一个加数
          ADC      AL, [BX]                  ; 相加
          DAA                               ; 十进制调整
          STOSB                               ; 结果送入 DI 所指单元
          INC      BX
          RET                                ; 返回
AAA       ENDP
CODE      ENDS                             ; 程序段结束
```

END                      START    ; 程序结束

【例 3-17】 确定字符串的长度。从头搜索字符串的结束标志 (0DH)，统计搜索的字符个数。

```
DATA    SEGMENT
STRING  DB    "ABCDEFGHIJ", 0DH
COUNT  EQU  $-STRING
LL      DB    ?
DATA    ENDS
STACK   SEGMENT PARA STACK "STACK"
        DB    100 DUP (?)
STACK   ENDS
CODE    SEGMENT
        ASSUME CS: CODE, DS: DATA, ES: DATA, SS: STACK
START   MOV    AX, DATA
        MOV    DS, AX
        MOV    ES, AX
        LEA    DI, STRING      ; 置被搜索串地址指针
        MOV    DL, 0           ; 置串长度初值为 0
        MOV    AL, 0DH
        MOV    CX, COUNT+10    ; 置循环次数大于串长度
AGAIN:  SCASB
        JE     DONE            ; 找到结束标志, 停止
        INC    DL
        DEC    CX              ; 循环次数减 1
        JNE    AGAIN          ; 规定的循环次数未完, 循环
        JMP    ERROR          ; 转至出错处理程序
DONE:   MOV    LL, DL
        HLT
        CODE   ENDS
        END    START
```

【例 3-18】 加偶校验到 ASCII 码。若有一个 ASCII 字符串，它们的起始地址放在单元 STRING 内，要求从串中取出每一个字符，检查其中包含的“1”的个数，若已为偶数，则它的最高有效位置“0”；否则，最高有效位置“1”后，送回。

程序流程图如图 3-7 所示。相应的程序为：

```
NAME    PARITY_CHECK
DATA    SEGMENT
STRING  DB    '1234567890'
```

```

COUNT    EQU        $-STRING
DATA      ENDS
STACK     SEGMENT    PARA    STACK 'STACK'
          DB          100 DUP (?)
STACK     ENDS
CODE      SEGMENT
          ASSUME      CS: CODE, DS: DATA,
                     ES: DATA, SS: STACK
START     MOV         AX, DATA
          MOV         DS, AX
          MOV         ES, AX
          LEA         SI, STRING
          MOV         CX, COUNT
AGAIN:    LODSB
          AND         AL, AL
          JPE        NEXT
          OR          AL, 80H
          MOV         [SI-1], AL
NEXT:     DEC         CX
          JNZ        AGAIN
          HLT
CODE      ENDS
          END         START
    
```

【例 3-19】 ASCII 码到十进制（BCD）数的转换。  
若有一输入 ADCII 码串（长度在串中的第一字节），要把其中的数码取出来，转换为未组合的 BCD 码，放至另一缓冲区中，并统计数码串的长度，放入此缓冲区的第一个字节。

程序流程图如图 3-8 所示。相应的程序为：

```

DATA      SEGMENT
LL         DB          10
STRING     DB          '123ASDFGKL'
BUFFER     DB          ?
          DB          10 DUP (?)
DATA      ENDS
STACK     SEGMENT    PARA    STACK 'STACK'
          DB          100 DUP (?)
STACK     ENDS
    
```

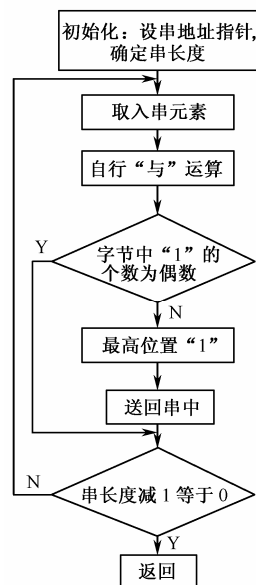


图 3-7 加偶校验到 ASCII 字符串的程序流程图

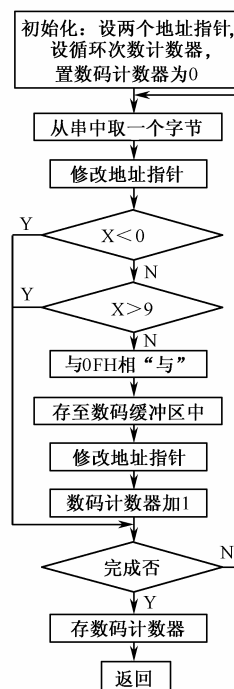


图 3-8 把 ASCII 码转换为 BCD 的程序流程图

```

CODE    SEGMENT
        ASSUME  CS: CODE, DS: DATA,
                ES: DATA, SS: STACK
START    MOV     AX, DATA
        MOV     DS, AX
        MOV     ES, AX
        MOV     CL, LL
        MOV     CH, 0
        LEA     SI, STRING
        LEA     DI, BUFFER
        INC     DI
        MOV     DL, 0
AGAIN:   LODSB
        CMP     AL, '0'
        JL      NEXT
        CMP     AL, '9'
        JG      NEXT
        AND     AL, 0FH
        STOSB
        INC     DL
NEXT:    LOOP    AGAIN
        MOV     BUFFER, DL
        HLT
CODE    ENDS
        END     START
    
```

【例 3-20】 二进制数到 ASCII 的转换。把在内存变量 **NUMBER** 中的 16 位二进制数，每一位转换为相应的 ASCII 码，存入串变量 **STRING** 中。

程序流程图如图 3-9 所示。相应的程序为：

```

DATA    SEGMENT
NUM      DW      4F78H
STRING   DW      16 DUP (?)
DATA     ENDS
STACK   SEGMENT PARA STACK 'STACK'
        DB      100 DUP (?)
STACK   ENDS
CODE    SEGMENT
        ASSUME  CS: CODE, DS: DATA,
    
```

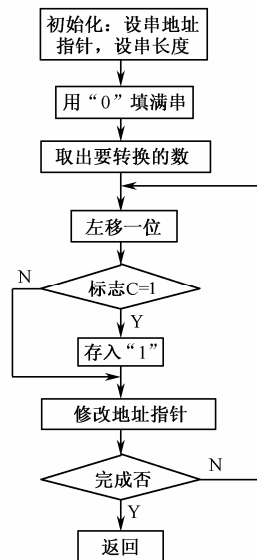


图 3-9 把二进制位串的每一位转换为 ASCII 码的程序流程图

```

                                ES: DATA, SS: STACK
START    MOV    AX, DATA
          MOV    DS, AX
          MOV    ES, AX
          LEA    DI, STRING
          MOV    CX, LENGTH STRING
          PUSH   DI
          PUSH   CX
          MOV    AL, 30H          ; 使缓冲区全置为 0
          REP    STOSB
          POP    CX
          POP    DI
          MOV    AL, 31H
          MOV    BX, NUM
AGAIN:    RCL    BX, 1            ; 左移 BX, 把相应位送入 C 标志
          JNC    NEXT            ; 若为 0 则转至 NEXT
          MOV    [DI], AL        ; 若为 1, 则把 1 置入缓冲区
NEXT:     INC    DI
          LOOP   AGAIN
          HLT
CODE      ENDS
          END    START

```

【例 3-21】 在 CRT 上连续输出字符 0~9。DOS 的功能调用 2 就是向 CRT 输出一个字符的子程序，它要求把要输出的字符的 ASCII 码送至寄存器 DL。即：

```

MOV    DL, OUTPUT_CHAR
MOV    AH, 2
INT    21H

```

为了使输出的字符之间有间隔，在每一循环中，输出一个 0~9 的字符和一个空格。要输出 0~9，只要使一个寄存器（程序中为 BL）的初值为 0，每循环一次使其增量。为了保证是十进制数，增量后要用 DAA 指令调整；为了保证始终是一位十进制数，用 AND 0FH 指令屏蔽掉高 4 位。

其程序流程图如图 3-10 所示。相应的程序为：

```

STACK    SEGMENT PARA STACK 'STACK'
          DB    100 DUP (?)
STACK    ENDS

```

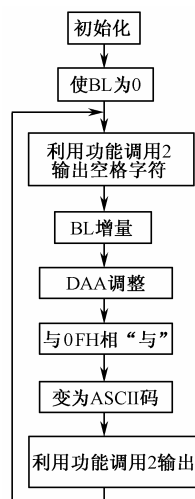


图 3-10 在 CRT 上输出 0~9 的程序流程图

```

CODE    SEGMENT
        ASSUME  CS: CODE, SS: STACK
START   MOV     BL, 0
        PUSH    BX
GOON:    MOV     DL, 20H          ; 把空格字符→DL
        MOV     AH, 2
        INT     21H             ; 输出空格字符
        POP     BX
        MOV     AL, BL
        INC     AL
        DAA                      ; 增量后进行十进制数调整
        AND     AL, 0FH
        MOV     BL, AL
        PUSH    BX
        OR      AL, 30H          ; 转换为 ASCII 码
        MOV     DL, AL
        MOV     AH, 2
        INT     21H             ; 输出一个 0~9 之间的字符
        MOV     CX, 0FFFFH      ; 为便于观察, 插入一定的延时
AGAIN:   DEC     CX
        JNE     AGAIN
        JMP     GOON
CODE    ENDS
        END     START

```

## 习题

1. 指出下列指令中的源操作数和目的操作数的寻址方式。

- (1) MOV AL, 08H
- (2) MOV AX, DATA[SI]
- (3) MOV AX, MASK[BP][SI]
- (4) MOV BX, AX
- (5) MOV BX, [4000H]

2. 试述下列两条指令的区别。

```

MOV AX, 4000H
MOV AX, [4000H]

```

3. 写出以下指令中内存操作数所在的地址。

- (1) MOV AL, [BX+8]
- (2) MOV AX, DATA[BP][SI]

(3) MOV DX, [2000H]

(4) MOV AX, [BP]

(5) MOV AX, [BX]

4. 判断下列指令是否正确：

(1) MOV BL, AX

(2) MOV AL, 2000H

(3) MOV CL, AX

(4) MOV DI, AL

(5) MOV DX, 2000H

5. 内存寻址中段寄存器与偏移地址的对应关系如何？

6. 操作数的寻址方式有哪些？举例说明。

7. 指令系统中主要有哪些类别的指令？

8. 已知两个无符号数 AL=20H, BL=10H。当执行“CMP AL, BL”后，比较结果如何？影响哪几个标志位？

9. 在 DATA 之下存放 100 个无符号 8 位数。试编写程序找出其中最小的数，并将其放在 TAB 中。

10. 试编写程序：将 BUFFER 中的一个 8 位二进制数转换为 ASCII 码，并按位数高低顺序存放在 HETB 中。

11. 16 位二进制数存放在 DATA 的连续两个单元中。试编写程序：求两个数的和，将结果送入 TAB 中。

# 第4章

## 总线技术

### 教学目的和要求

总线是连接计算机各部件或计算机之间的公共信号通道。本章简要介绍总线的分类及特性。重点掌握总线的基本概念，要求一般地了解总线在计算机中的应用方法、总线信号的分类及总线的标准化。



总线是一组信号线的集合，是一种在各模块间传送信息的公共通路。在微型计算机系统中，利用总线实现芯片内部、印制电路板各部件之间、机箱内各插件板之间、主机与外部设备之间或系统与系统之间的连接与通信。总线是构成微型计算机应用系统的重要技术，总线设计好坏会直接影响整个微机系统的性能、可靠性、可扩展性和可升级性。由于总线在系统中的重要地位，微机系统的设计和开发人员，先后推出许多种总线标准。

总线标准一般以两种方式推出：一种是某公司在开发自己的微机系统时所采用的一种总线，而其他兼容机厂商都按其公布的总线规范开发相配套的产品并进入市场。这种总线被国际工业界广泛支持，有的还被国际标准化组织加以承认并授予标准代号。另一种是由国际权威机构或多家大公司联合制订的总线标准。前一种先有产品后有标准，如 IBM PC/AT 上使用的 ISA 总线。后者先有标准后有产品。随着微机系统的更新换代，有的总线仍在发展完善，如 STD SUB，MULTI BUS 等，而有的就逐渐衰亡甚至被淘汰。本章将简要介绍几种流行的标准总线。

## 4.1 总线的基本概念

### 1. 总线分类

总线按系统传输信息的不同可分为三类：数据总线、地址总线和控制总线。

数据总线用来在各功能部件之间传输数据信息，它是双向的传输总线，其位数与机器字长、存储字长有关，一般为 8、16、32 及 64 位。数据总线的条数称为数据总线的宽度。

地址总线主要用来指出数据总线上源数据或目的数据在主存储单元或 I/O 端口的地址。地址总线为单向传输，其宽度一般为 16、20、24、32 及 64 位。

控制总线是用来传输各种控制信号的传输线。通常一条控制信号线的信号传输是单向的，当然也有双向的。控制总线还可以起到监视各部件状态的作用，例如，查询某个设备是否处于“忙”或“闲”的状态。常见的控制信号有：时钟、复位、总线请求、总线允许、中断请求、中断确认、存储器写、存储器读、I/O 写、I/O 读、数据确认等。

大多数微机采用了分层次的多总线结构。总线按在系统的不同层次位置上分类，可分为以下 4 类。

#### （1）片内总线

片内总线是指一些大规模集成电路内部的总线，是用来连接各功能部件的信息通路。例如，CPU 芯片中的内部总线，它是 ALU 寄存器和控制器之间的信息通路。片内总线根据其功能又被分为地址总线、数据总线和控制总线。这种总线是由微处理机芯片生产厂家设计的，对系统的设计者和用户来说关系不大。但是随着微电子学的发展，出现了 ASIC（专用集成电路）技术，用户可以按自己的要求借助 CAD 技术，设计自己的专用芯片，在这种情况下，用户就必须掌握片内总线技术。

由于片内总线所连接的部件都在一个硅片上，追求高速度是它的主要目标，所以器件级的总线都采用并行总线。同样为了提高速度，克服一组总线上同一时刻只能有两个部件通信所造成的限制，还采取了多总线的措施。使芯片中可以有一个以上的通信同时进行，实现片内几个部件并行工作，大大地提高了芯片的工作速率。例如，Pentium CPU 内的 11 个大

部件就可以同时操作，使它对指令的处理速度得到极大的提高。

### (2) 微处理器总线

微处理器总线或称处理器总线、主板局部总线、元件级总线，它是指在印制电路板上连接各芯片的公共通路。它可能是一台单板计算机或是一块 CPU 主板上用于芯片一级的连接总线。它是微型机系统的重要总线。它一般是 CPU 芯片引脚的延伸，与 CPU 的关系密切。但当板内芯片较多时，往往需增加锁存、驱动等电路，以提高驱动能力。

### (3) 系统总线

系统总线又称为内总线、板级总线，它用于微型机系统各插件板之间的连接，是微型机系统最重要的一种总线。一般谈到微型机总线，指的就是这一种总线。有的系统总线是局部总线经过重新驱动和扩展而成的，其性能与某种 CPU 有关。但有不少系统总线并不依赖于某种型号的 CPU，可为多种型号 CPU 及其配套芯片所使用，在用各种插件板来组成或扩充微型机系统时，就要选用恰当的系统总线，并按总线的规定来制作这些插件板。系统总线一般都做成多个插槽的形式，各插槽相同的引脚都连到一起，总线就连到这些引脚上。

### (4) 外部总线

外部总线又称为通信总线，它用于微机系统之间、微机系统与仪器或其他设备之间的通信通道。这种总线数据传输方式可以是并行（如打印机）或串行的，数据传输速率比内总线低。不同的应用场合有不同的总线标准。例如，串行通信的 RS-232C 总线，用于硬磁盘接口的 IDE、SCSI，用于连接仪器仪表的 IEEE-488 等总线。这种总线并非微机专有，一般是利用工业领域已有的标准。

## 2. 总线标准的基本内容

### (1) 物理特性

物理特性指的是总线物理连接的方式，包括总线的插头、插座的尺寸及形状，总线的根数和引脚是如何排列的等。

### (2) 功能特性

功能特性是确定引脚名称与功能，以及其相互作用的协议。从功能上看，总线分为地址总线、数据总线、控制总线、备用线、电源和地线。

地址总线的宽度指明了总线能够直接访问存储器或外部设备的地址范围。数据总线的宽度指明了访问一次存储器或外部设备最多能够交换数据的位数。控制总线一般包括 CPU 与外界联系的各种控制命令，如读/写控制逻辑线、时钟线和电源线等；中断机制；总线主控仲裁；应用逻辑，如握手联络线，复位、自启动、休眠维护等。备用线留做功能扩充和用户的特殊要求使用。电源和地线决定了总线使用的电源种类及地线分布和用法。

### (3) 电气特性

电气特性规定每一根线上信号的传输速率的设定、驱动能力的限制、信号电平的规定、时序的安排，以及信息格式的约定等。一般规定送入 CPU 的信号叫输入信号，从 CPU 送出的信号叫输出信号。

## 3. 总线的操作过程

微机系统中的总线操作，包括从 CPU 把数据写入存储器、从存储器把数据读到 CPU、

从 CPU 把数据写入输出端口、从输入端口把数据读到 CPU、CPU 中断操作、直接存储器存取操作等，本质上都是通过总线进行的信息交换，统称为总线操作。

总线操作是在主控模块的控制下进行的，主控模块是具有控制总线能力的模块（如 CPU 及 DMA 模块）。总线从属模块没有控制总线的能力，它可对总线上传来的信号进行地址译码，并且接受和执行总线主控模块的命令信号。在同一时刻，总线上只能允许一对模块进行信息交换。当有多个模块都要使用总线进行信息传送时，只能采用分时方式，一个接一个地轮流交替使用总线，即将总线时间分成很多段，每段时间可以完成模块之间一次完整的信息交换，通常称之为一个数据传送周期或一个总线操作周期。为完成一次总线操作周期，一般要分成 4 个阶段。

#### （1）申请阶段

当系统总线上有多个主控模块时，由需要使用总线的主控模块向总线仲裁机构提出使用总线的申请，经总线仲裁机构判别确定，把下一个总线传输周期的总线控制权授给哪个申请者。

#### （2）寻址阶段

取得总线使用权的主控模块通过总线发出本次访问的从属模块的地址及有关命令，以启动参与本次操作的从属模块。

#### （3）传数阶段

主控模块和从属模块之间进行数据传输，数据由源模块发出经数据总线流入目的模块。在进行读操作时，源模块就是存储器或输入 / 输出接口，而目的模块则是总线主控模块。在进行写操作时，源模块就是总线主控模块，而目的模块则是存储器或输入 / 输出接口。

#### （4）结束阶段

主从模块的有关信息均从系统总线上撤除，让出总线，以便其他模块能继续使用。

### 4. 总线的数据传输方式

主控模块和从属模块之间的数据传输方式，可分为以下 3 种，它们各有优缺点。

#### （1）同步传送

同步传送时采用一个“系统时钟”作为各模块动作的基准时间。模块间通过总线进行一次传送所需的时间是固定的，其中每一步骤的起止时刻，也都有严格的规定，都以系统时钟来统一步伐。很多微机系统的基本传送方式都是同步传送。

同步传送方式的存储器读总线周期时序如图 4-1 所示，其具体操作如下：在  $T_1$  的前沿，CPU 向地址总线发出访问从模块（存储器单元）的地址；在  $T_2$  内，CPU 发出读命令  $\overline{RD}$  有效；在  $T_3$  内，被选中的存储器单元把数据送到数据总线上；在  $T_4$  内，存储器撤除数据，CPU 撤除地址和读命令。

同步传送的主要优点是简单，数据传送由单一信号控制。由于采用了公共时钟，每个功能模块什么时候发送或接收信息都由统一时钟规定，因此，同步传送具有较高的传输频率。同步传送适用于总线长度较短、各功能模块存取时间比较接近的情况。这是因为同步方式对任何两个功能模块的通信都给予同样的时间安排。由于同步总线必须按最慢的模块来设计公共时钟，当各功能模块存取时间相差很大时，会大大降低总线效率。

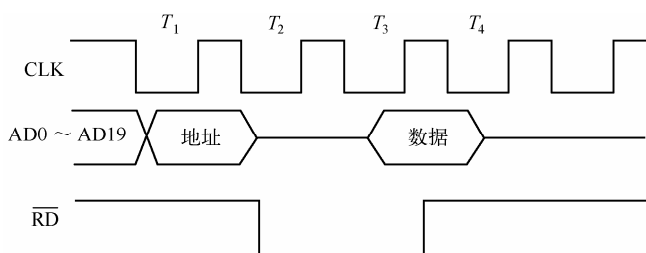


图 4-1 同步时序

### (2) 异步传送

同步传送要求总线上的各主从模块操作速度要严格匹配,对于具有不同存取时间的各种模块,是不适宜采用同步传送方式的。为了对高速模块能具有高速操作,而对低速模块能具有低速操作,从而对不同的模块具有不同的操作时间,可采用异步传送方式。

异步传送方式采取“应答式”传输技术,用“请求”和“应答”信号线来协调传送过程,而不依赖于公共时钟信号。它可以根据模块的速度自动调整响应的的时间,任何类型的模块都不需要考虑速度问题,从而避免同步传送方式的上述缺点。

异步传送的读命令时序如图 4-2 所示。对读操作,主模块在地址建立以后,送出低电平有效的读请求信号①。总线上的所有从模块各自进行地址译码和判断选择,被选中的模块响应这一请求,将数据读出放在总线上②。从模块此刻将应答线变为低电平有效,标识已将主模块所需的数据放在数据总线上,等待主模块去读取③。主模块在检测到应答线有效后,就将读请求信号由低电平变为高电平。主模块利用这个由低到高的跳变将数据锁存,完成数据的读取,并表示读请求信号已撤除。然后撤除地址和数据④。最后撤除应答线⑤。

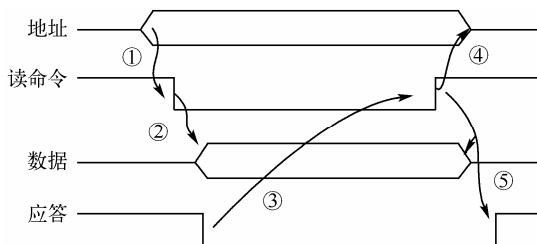


图 4-2 异步时序

### (3) 半同步传送

半同步传送方式是前两种方式的折中。从总体上看,它是一个同步系统,仍用系统时钟来定时,利用某一脉冲的前沿或后沿判断某一信号的状态,或控制某一信号的产生或消失,使传送操作与时钟同步。但是,它又不像同步传送那样传送周期固定,对于慢速的从模块,其传送周期可延长时钟脉冲周期的整数倍。其方法是增加一条“等待(WAIT)”或“准备就绪(REDAY)”信号线。WAIT 信号线有效(或 REDAY 无效)时,反映选中的从模块未准备好,系统就自动地将传送周期延长一个时钟周期(强制主模块等待),此状态时钟后每个时钟均进行检测,直至检测到 WAIT 信号线无效,才不再延长传送周期。这又像异步传送那样传送周期视从模块的速度而异,允许不同速度的模块和谐地一起工作。但这个 WAIT 信

号不是互锁的, 只是单方向的状态传递, 这又是不同之处。现在采用半同步传送方式的微机系统较多。

## 4.2 IBM PC总线

IBM PC 的 I/O 通道是系统总线的扩充, IBM PC/XT 个人计算机上采用的微型计算机总线, 亦称 XT 总线。IBM PC 对 I/O 通道上的信号名称、性质、方向时序、引脚排列都有明确的要求, 以便厂家和用户制作与之匹配的插件板。这一规范亦被称为 IBM PC 总线标准。

PC/XT 系统的主板上有五个功能区, 分别是处理器子系统、ROM 子系统、RAM 子系统、各种 I/O 适配器 (模板) 插槽和 I/O 通道支持部件。

PC/XT 中有 8 个 62 引脚的扩展槽, 这 8 个扩展槽是实现了对系统进行扩展的手段, 扩展槽上可以插入不同功能的插件板, 用来扩充系统的功能, PC/XT 可以通过在 I/O 扩展槽中插入相应的适配器来连接各种外设。

与扩展槽相连的 62 根线组成 IBM PC/XT 系统总线。62 根总线中包括 8 位双向数据总线、20 位地址总线、6 根中断请求信号线、3 组 DMA 通道控制线、存储器和 I/O 读/写控制线、存储器刷新控制和时钟信号线、通道检验线、四种电源线及地线。这些引线均接在 62 插脚的插座上, 双列插脚分别为  $A_1 \sim A_{31}$  (A 面) 和  $B_1 \sim B_{31}$  (B 面)。插座的引脚间距为 100mil (密尔)。

62 条信号线按功能可分为如下 5 类:

### 1. 数据线 (8 根)

$D_7 \sim D_0$  双向数据线, 是 CPU 与存储器和 I/O 设备之间交换信息的通路,  $D_0$  是最低有效位,  $D_7$  是最高有效位。

### 2. 地址线 (20 根)

$A_{19} \sim A_0$  地址总线, 是输出信号,  $A_0$  为最低位,  $A_{19}$  为最高位。如果传送 I/O 地址, 则  $A_{19} \sim A_{16}$  无效。地址可由 CPU 产生, 也可以由 DMA 产生。

### 3. 控制线 (21 根)

(1)  $\overline{ALE}$ : 地址锁存, 输出信号,  $ALE$  有效时为高电平, 当其跌落为无效低电平时, 它的下跳沿把来自 CPU 的地址锁存。 $ALE$  在 DMA 周期中无效。

(2)  $IRQ_2 \sim IRQ_7$ : 中断请求输入信号, 这些信号都是由 I/O 设备直接送到中断控制器 8259A 的。8259A 的  $IRQ_0$  和  $IRQ_1$  被系统板占用。ROM 中的 BIOS 程序将 8259A 控制器初始化成  $IRQ_2$  的优先级最高, 而  $IRQ_7$  的优先级最低。如果这一级未被屏蔽, 信号的上升沿就产生中断请求。一旦产生上升沿请求, 它就保持有效电平, 直到 CPU 响应为止。

(3)  $\overline{IOR}$ : I/O 读控制输出信号, 低电平有效。用来向 I/O 端口指明当前启动的总线周期是 I/O 端口读周期, 还指明地址总线上的地址是一个 I/O 端口地址。

(4)  $\overline{IOW}$ : I/O 写控制, 它是输出信号, 低电平有效。指明在地址总线提供的是一个 I/O 端口地址, 并指明数据总线上有一个要写至该 I/O 端口的数据。

(5)  $\overline{MEMR}$ : 存储器读控制, 输出信号, 低电平有效。该信号有效时, 表明地址总

线上有一个存储单元地址，并将该地址所选中的存储单元中的数据读到数据总线上。

(6)  $\overline{\text{MEMW}}$ ：存储器写控制，输出信号，低电平有效。该信号有效时，表明地址线上的地址是存储单元地址，并将数据总线上的数据写入所选中的单元中。

(7)  $\text{DRQ}_1 \sim \text{DRQ}_3$ ：DMA 请求，输入信号，高电平有效。这几条信号线直接连至系统板上的 8237-5 DMA 控制器，这些信号是由外设接口发出的。IBM PC 的 ROM BIOS 中程序将该 DMA 控制器初始化成  $\text{DRQ}_1$  的优先级最高， $\text{DRQ}_3$  的优先级最低。当某一通道有 DMA 请求时，对应的 DRQ 为高电平，该信号一直保持到相应的  $\overline{\text{DACK}}$  为低电平为止。

(8)  $\overline{\text{DACK}}_0 \sim \overline{\text{DACK}}_3$ ：DMA 请求响应，输出信号，低电平有效。这些信号是由 DMA 控制器送往外设接口的 DMA 响应信号，表明对应的 DRQ 请求已被接受，DMA 控制器将要占用并开始处理所请求的 DMA 周期。系统总线上没有对应的  $\text{DRQ}_0$ ，因此  $\overline{\text{DACK}}_0$  的发出仅表明当前的 DMA 周期是一个可以用来刷新系统动态存储器的虚拟读周期。

(9) AEN：地址允许输出信号，高电平有效。它是由 DMA 控制器 8237-5 输出的，该信号有效，用来切断 CPU 的控制，即使 CPU 的地址、数据和控制总线与系统总线的联系中断，而使 DMA 控制器掌握地址总线和数据总线，允许 DMA 传送。

(10) TC：计数结束输出信号，高电平有效。它由 DMA 控制器 8237-5 输出，表明某个 DMA 通道已达到其程序预置的传送周期数，由 TC 线输出。该信号通常用来结束一次 DMA 数据块传送。由于这个信号在四个 DMA 通道中的任何一个达到某终点值时都发出，因此，接口逻辑应该将 TC 和  $\overline{\text{DACK}}$  相“与”，才能得出特定通道的 TC。

(11) RESET DRV：系统复位输出信号，高电平有效。在系统电源接通后，保持其为高电平，一直到所有电平都达到其规定的工作范围后才失效。这一信号一般用来向与总线连接的接口逻辑或 I/O 设备提供复位信号，使它们在系统操作之前达到其已知的状态。

#### 4. 状态线（2 根）

(1) I/O CH RDY：I/O 通道准备就绪输入信号，用来延长总线周期。对于一些较慢的 I/O 设备或存储器，可以通过使该信号变为低电平，从而使 CPU 或 DMA 加入等待状态，于是延长总线周期。

(2) I/O CH CK：I/O 通道检查输入信号，此信号用来向 CPU 提供关于 I/O 通道上的设备或存储器的奇/偶校验信息。当它为低电平时，表示奇/偶校验错。

#### 5. 辅助线和电源线（11 根）

(1) OSC：晶体振荡输出信号，频率为 14.318 18MHz，周期约为 70ns，占空比为 50%。所有其他的时序信号均由此信号产生。

(2) CLK：系统时钟输出信号，它是由 OSC 三分频得到的，频率为 4.77MHz，周期为 210ns。占空比为 33%，其中高电平 70ns。典型总线周期为四个时钟周期，即约为 840ns。

(3)  $\overline{\text{CARDSLCTD}}$ ：插件板选中输入信号，低电平有效。有效时表示扩展槽  $J_8$  中的扩展板被选中。

### 4.3 ISA总线

ISA（Industry Standard Architecture，工业标准总线）总线是PC中最基本的总线，是在8位PC总线的基础上扩展而成的16位的总线体系结构，其数据宽度为16位，地址宽度为24位，工作频率为8MHz，最大数据传输速率为5MB/s。它适用于对速度要求不太高的板卡和外设中，如串行口、并行口、声音卡等。在PC中，ISA总线插槽由两组插座组成，其中长的一组用于插8位的与ISA相兼容的板卡。ISA插槽通常为黑色的。

ISA总线是在PC/XT总线基础上增加了1个36线插座而形成的。实际上，与PC/XT总线相比，ISA总线不仅增加了数据线宽度和寻址空间，还加强了中断处理（新增了7个中断级别）和DMA（新增了3个DMA）传输能力，并且具备了一定的多主功能。故ISA（AT）总线特别适合于控制外设和进行数据通信的功能模块。

ISA板卡有长短两个插口，长插口有62个引脚，以A<sub>1</sub>~A<sub>31</sub>和B<sub>1</sub>~B<sub>31</sub>分列于板的两面；短插口有36个引脚，以C<sub>1</sub>~C<sub>18</sub>和D<sub>1</sub>~D<sub>18</sub>分列于板的两面。ISA总线插座具有98个引脚，包括接地和电源引脚10个、数据线引脚16个、地址线引脚27个、各种控制信号引脚45个。

表4-1给出了ISA总线I/O端口地址的典型使用。

表 4-1 ISA 总线 I/O 端口地址的典型使用

I/O 口地址 (十六进制数)	设备 (系统板上的外围电路)	I/O 口地址 (十六进制数)	设备 (适配器上的外围电路)
000~01F	DMA 控制器 1, 8237A-5	360~36F	保留
020~03F	中断控制器 1, 8259A (主)	378~37F	并行打印机口 2
040~05F	定时器, 8253	380~38F	SDLC, 双同步 2
060~06F	8042 (键盘接口处理器) 的 PB 口	3A0~3AF	双同步 1
070~07F	实时时钟, NMI 屏蔽寄存器	3B0~3BF	单色显示器和打印机适配器
080~09F	DMA 页面寄存器	3C0~3CF	保留
0A0~0BF	中断控制器 2, 8259A (从)	3D0~3DF	彩色/图形显示适配器
0C0~0DF	DMA 控制器 2, 8237A	3F0~3F7	软磁盘控制器
1F0~1F8	硬磁盘	3F8~3FF	串行口 1
200~207	游戏 I/O 口		
278~27F	串行口 2		
300~31F	样卡		

### 4.4 PCI总线

随着微型计算机技术的广泛应用和不断发展，无论是办公自动化还是工业应用，对微型计算机性能的要求都越来越高。在CPU从80286发展到80386、80486及目前的Pentium水平的情况下，其数据宽度及工作频率也在不断提高。可是，传统的ISA总线、EISA总线及VESA总线严重地限制了新型芯片水平的发挥和利用，这就促使微型计算机总线技术也要日趋完善，不断推出新标准。而PCI局部总线既符合当今的技术要求，又能满足未来的

需要,是一种较好的局部总线标准。PCI 高性能、高效率、与现有标准强大的兼容性和充裕的开发潜力,是其他总线所不及的,因而成为开发当今和未来微型计算机的重要基础。

#### 4.4.1 PCI总线的特点

PCI (Peripheral Component Interconnect) 总线,即外围部件互连总线,是 1991 年下半年由 Intel 公司首先提出的一种局部总线标准,主要用于高档微机的主板,该总线插槽为白色。

PCI 是先进的高性能局部总线,可同时支持多组外部设备。PCI 局部总线不受处理器限制,为中央处理器及高速外部设备提供数据传输通道,进行总线之间的数据传输的调度管理。PCI 采用高度综合化的局部总线结构,以确保微型计算机中各部件、附加卡及系统之间的可靠运行,并能完全兼容现有的 ISA/EISA/MicroChannel 扩充总线。PCI 总线的特点如下。

##### (1) 高性能

PCI 是一套整体的系统解决方案,不仅可以提高网络接口卡和硬盘等设备的性能,还能满足图形及各种高速外部设备的要求。PCI 总线的时钟频率为 33 MHz 和 66 MHz,传输速率从 132 Mb/s (33MHz 时钟、32 位数据通路)可升级到 528 Mb/s (66MHz、64 位数据通路),满足了当前及以后相当一段时期内 PC 传输速率的要求。

##### (2) 猝发传输模式

PCI 支持猝发(或称成组传送)数据传输模式。关于猝发传输模式已在第 2 章讲述过,即如果被传送的数据在内存中连续存放,则在访问这一组连续数据时,只有在传送第一个数据时需要两个时钟周期,第一个时钟周期给出地址,第二个时钟周期传送数据;而传送其后的连续数据时,不必每次都给出地址(即每次只需将地址自动加 1),只要一个时钟周期便可传送一个数据。PCI 可以支持猝发读取和猝发写入,这对使用高性能图形设备来说是非常重要的。

##### (3) 不受微处理器限制

PCI 总线以其独特的中间缓冲器方式,使 PCI 总线部件和插件接口相对于微处理器是独立的,PCI 总线支持所有的目前和将来不同结构的微处理器,并将微处理器系统与外部设备分开。一般来说,在微处理器总线上增加更多的设备或部件,只会降低性能和可靠程度。而有了这种缓冲器的设计方式,用户可随意增添外部设备,以扩充微机系统而不必担心在不同时钟频率下会导致性能的下降。这种独立于微处理器的总线设计还可保证不会因微处理器技术的变化而导致其他互连外设系统的设计变得过时,因此具有相对长的生命周期。

##### (4) 采用总线主控和同步操作

PCI 总线所具有的主控和同步操作功能有利于提高 PCI 总线性能。它允许许多处理机系统中的任何一个微处理机都可以成为总线主控设备,对总线操作进行控制,体现了高度接纳设备的灵活性。总线主控是大多数总线都具有的功能,目的是让任何一个具有处理能力的外部设备暂时接管总线,以加速执行高吞吐量、高优先级的任务。PCI 总线独特的同步操作功能可保证微处理器能够与这些总线主控器同时工作,而不必等待后者任务的完成。



#### （5）减少存取延迟

支持 PCI 总线的设备，减少存取延迟，能够大幅度减少外部设备取得总线控制权所需的时间，以保证数据传输的畅通。

#### （6）适用于各种机型

PCI 局部总线不仅适用于各种桌面式微机，而且还适用于便携式微机以及服务器等。通过支持 3.3V 的电源环境，使 PCI 局部总线应用在便携式微机中，可减小微机的体积，并增加其功能。

在服务器环境下，PCI 总线具有支持分级式外部设备的特性，可使一个 PCI 界面支持一组级联的 PCI 局部总线，也可以设置为多组 PCI 总线的服务器增添额外的扩展插槽，提供更多的 I/O 接口，并将高带宽与低带宽的数据分隔开来。

#### （7）兼容性强

PCI 总线与 ISA、EISA 及 VESA 等总线完全兼容，方便用户选用外部设备。优良的软件兼容性，即 PCI 部件可完全兼容现有的驱动程序和应用程序，设备驱动程序可被移植到各类平台上。

它支持 64 位地址/数据多路复用，PCI 插槽能同时插 32 位和 64 位插卡，所以，32 位和 64 位外部设备是在用户不知不觉中工作的，因而它们之间的相互通信对用户来说是透明的，从而达到真正的前后兼容。

它支持 5 V 和 3.3 V 两种扩充插件卡，可以从 5 V 向 3.3 V 进行平滑的系统转换。PCI 总线上装有一个很小的断路键，使用户在插卡时不会导致在系统主板上有不同的电压电源。

它还提供了自动配置（即插即用）功能，保证了用户在安装一个新的添加卡时，无须手工调整 DIP 开关、跳线（跨接线）和选择的中断。配置软件会自动选择未被使用的地址和中断，以解决可能出现的冲突问题。

#### （8）低成本、高效益

PCI 的芯片采用超大规模集成电路，节省布线空间，为微机的小型化和多功能化提供了良好的条件。PCI 部件采用地址/数据线复用，使 PCI 部件连接其他部件的引脚数减至 50 以下。PCI 总线引脚线的每两个信号之间都安排了一个地线，以减少信号间的相互干扰，以及音频信号的散射问题。PCI 到 ISA/EISA 的转换由芯片厂家提供，减少了用户的开发成本。

PCI 总线主要性能指标如下：

- （1）PCI 总线时钟频率为 33.3MHz 或 66.6MHz（可达 133MHz 及更高）。
- （2）总线数据宽度为 32 位或 64 位。
- （3）传输速率为 133MB/s（32 位）或 266MB/s（64 位）。
- （4）支持 64 位寻址。
- （5）适应于 5V 和 3.3V 两种电源环境。

### 4.4.2 PCI总线的系统结构

图 4-3 给出了 PCI 总线示意图，由图可以看出 PCI 总线是一种位于微处理器总线与系统总线之间的一种夹层总线。这就意味着 PCI 总线的控制器（或称桥、桥接器）位于 CPU 和系统总线之间。也就是说，任何一种 CPU 都可以使用 PCI 总线。对总线连线进行标准化处

理, 可以使 CPU 总线免受各种约束。扩展总线桥 (或称标准总线桥) 的设置是为了能在 PCI 总线上接出一条系统总线, 如 ISA、EISA 等总线, 从而可继续使用现有的 I/O 设备, 以增加 PCI 总线的兼容性和选择范围。

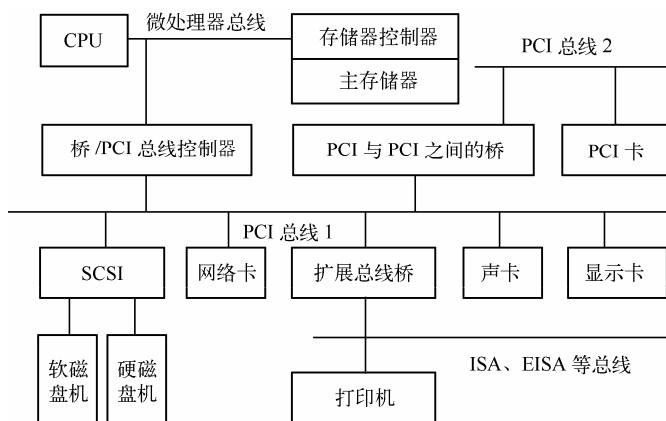


图 4-3 PCI 总线示意图

桥也叫桥连器, 是一个总线转换部件, 其功能是连接两条计算机总线, 使总线间相互通信。它可以把一条总线的地址空间映射到另一条总线的地址空间, 可以使系统中每一台总线主设备 (Master) 能看到同样的一份地址表。从整个存储系统看, 有了整体性统一的直接地址表, 可以大大简化编程模型。桥本身可以十分简单, 如只是加上信号的缓冲能力。也可以相当复杂, 如可以包括装拆数据分组以及有各类系统所规定的一些功能。在 PCI 规范中, 提出了三种桥的设计:

(1) 主桥, 就是 CPU 至 PCI 的桥。

(2) 标准总线桥, 即 PCI 至标准总线如 ISA、EISA、微通道之间的桥。例如, Intel 设计的 SATUNG 芯片组具有 ISA 标准总线桥路, 而为奔腾设计的 MERCURY 芯片组可选择 ISA 标准总线桥路, 也可选择 EISA 标准总线桥路。

(3) PCI 桥, 在 PCI 与 PCI 之间的桥。

其中, 主桥称为北桥 (North Bridge), 其他的桥称为南桥 (South Bridge)。

#### 4.4.3 PCI 总线信号

PCI 标准规定了三种适配卡配置, 分别对应不同的电源要求: 为传统 5V 系统规定的 5V 标准、为 3.3V 供电的移动系统规定的 3.3V 标准, 以及为采用两种电源工作的通用标准。

对于 5V PCI 标准连接器而言, 如果 PCI 适配卡仅支持 32 位操作, 则只用到管脚 B1/A1 到 B62/A62, 管脚 B63/A63 到 B94/A94 只用于 64 位 PCI 适配卡。

PCI 局部总线的信号线共有 100 根, 在一个 PCI 应用系统中, 有主设备和从设备。从设备至少需要 47 根信号线, 主设备则需要 49 根信号线。利用这些信号线可以处理数据、地址, 实现接口控制、仲裁及系统功能。下面按功能分组说明 5V PCI 标准连接器的引脚意义。

### 1. 系统接口信号

(1) **CLK**: PCI 系统总线时钟, 对于所有的 PCI 设备均为输入, 该信号频率为 PCI 总线的工作频率。除  $\overline{\text{RST}}$ 、 $\overline{\text{IRQB}}$ 、 $\overline{\text{IRQC}}$ 、 $\overline{\text{IRQD}}$  之外, 所有 PCI 的其他信号都在 CLK 的上升沿有效。

(2) **RST**: 复位信号, 用来使 PCI 专用的寄存器、序列器和相关信号复位到规定的初始状态。复位时, PCI 的全部输出信号一般都应驱动为第三态。 $\overline{\text{SERR}}$  信号为高阻状态,  $\overline{\text{SBD}}$  和  $\overline{\text{SDONE}}$  可驱动到低电平 (如果未提供三态输出)。 $\overline{\text{REQ}}$  和  $\overline{\text{GNT}}$  必须同时驱动为第三态, 不能在复位期间为高或低。为防止 AD、C/ $\overline{\text{BE}}$  及 PAR 在复位期间浮动, 可由中心设备将它们驱动到低电平, 但不能驱动为高电平。 $\overline{\text{RST}}$  和 CLK 可以不同步, 但要保证其撤消边沿没有反弹。当设备请求引导系统时, 将响应复位, 复位后响应系统引导。

### 2. 地址与数据接口信号

(1) **AD31~AD0**: 地址、数据多路复用的三态双向信号。在 PCI 总线传输时, 包含一个地址传送节拍和一个 (或多个) 数据传送节拍, 在  $\overline{\text{FRAME}}$  (帧周期信号) 有效时, 为地址传送节拍开始; 在  $\overline{\text{IRDY}}$  (主设备就绪信号) 和  $\overline{\text{TRDY}}$  (从设备就绪信号) 同时有效时, 为数据传送节拍。PCI 总线支持猝发方式的读/写功能。

(2) **C/ $\overline{\text{BE}}$ 3~C/ $\overline{\text{BE}}$ 0**: 总线命令/字节允许多路复用三态双向信号线。在地址传送节拍中, 这四条线是由主设备发向从设备的总线命令, 说明当前事务类型。在数据传送节拍内, 它们传输的是字节允许信号, 说明 AD31~AD0 上数据中哪些字节为有效数据。例如, C/ $\overline{\text{BE}}$ 0 对应第 0 字节, C/ $\overline{\text{BE}}$ 1 对应第 1 字节。

(3) **PAR**: AD31~AD0 和 C/ $\overline{\text{BE}}$ 3~C/ $\overline{\text{BE}}$ 0 的奇偶校验位 (偶校验), 它是三态双向信号, 可被所有 PCI 设备或模块使用。对于地址信号, 在地址传送节拍之后的一个时钟周期 PAR 稳定并有效; 对于数据信号, 在  $\overline{\text{IRDY}}$  或  $\overline{\text{TRDY}}$  有效之后的一个时钟周期 PAR 稳定并有效。一旦 PAR 有效, 就一直维持直到当前数据传送节拍完成之后的一个时钟周期为止。写操作时, PAR 由主设备驱动; 读操作时, PAR 由从设备驱动。

### 3. 接口控制信号

(1) **FRAME**: 帧周期信号, 由当前主设备驱动, 表示一次访问的开始和结束。 $\overline{\text{FRAME}}$  有效预示总线传输的开始, 在其有效期间, 说明数据传输继续进行;  $\overline{\text{FRAME}}$  无效说明为传输最后一个数据传送节拍。

(2) **IRDY**: 主设备准备好信号, 由系统主设备驱动。该信号有效说明引起本次传输的设备可以完成一个数据周期, 但要与  $\overline{\text{TRDY}}$  配合, 它们同时有效, 才能完成数据传输, 否则进入等待周期。在读周期, 该信号有效, 表示数据变量已出现在 AD31~AD0 上。在写周期, 该信号有效时, 表示从设备已做好接收数据的准备。

(3) **TRDY**: 从设备准备好信号, 由系统从设备驱动。该信号有效说明从设备已做好当前数据传输的准备工作, 可以进行相应的数据传输。同样, 该信号要与  $\overline{\text{IRDY}}$  配合使用, 二者同时有效, 才能完成数据传输。在写周期内, 该信号有效表示从设备已做好了接收数据的准备; 在读周期内, 该信号有效表示有效数据已提交到 AD31~AD0 上。同理,  $\overline{\text{IRDY}}$  和  $\overline{\text{TRDY}}$  的任何一个无效时都为等待周期。

(4)  $\overline{\text{STOP}}$ : 停止数据传送信号, 由从设备发出。当它有效时, 表示从设备要求主设备停止当前的数据传送。

(5)  $\overline{\text{LOCK}}$ : 锁定信号, 该信号有效, 表示驱动它的设备所进行的操作可能需要多次传输才能完成, 即此设备的操作将独占总线资源。而此时, 对于未被锁定的设备, 对它的非互斥访问仍然可以进行。 $\overline{\text{LOCK}}$  信号由 PCI 总线上发起数据传输的设备给出, 是根据它自己的约定并结合  $\overline{\text{GNT}}$  信号来完成的。即使有几个不同的设备在使用总线, 但对  $\overline{\text{LOCK}}$  信号的控制权只属于某个主设备。如果某一设备需要访问存储器, 那么访问存储器的操作必须实现锁定, 以便实现对该存储器的完全独占性访问。对于支持锁定的从设备, 必须能提供一个互斥访问块, 且该块不能小于 16 字节。由于主桥后面是系统存储器, 所以也能实现锁定。

(6)  $\overline{\text{DEVSEL}}$ : 设备选择信号, 该信号有效 (输出), 表示所译码的地址是在设备的地址范围内; 当作为输入信号时, 表示总线上某设备已被选中。

需要注意的是, 上述 (1) ~ (6) 信号和下面将要讲述的  $\overline{\text{PERR}}$ 、 $\overline{\text{REQ64}}$ 、 $\overline{\text{ACK64}}$  信号均表示持续的且低电平有效的三态信号, 这些信号在某一时刻只能属于一个主设备并被其驱动, 它从有效变为浮空 (高阻状态) 之前必须保证具有至少一个时钟周期的高电平状态。另一主设备要想驱动它, 至少要等待该信号的原有驱动者将其释放 (变为三态) 一个时钟周期之后才行。同时, 如果此类信号处于持续的非驱动状态时, 在有新的主设备驱动之前应采取上拉措施。并且, 该资源必须由中央资源提供。

(7)  $\overline{\text{IDSEL}}$ : 初始化设备选择输入信号。在配置读/写传输时, 作为片选信号。

#### 4. 仲裁信号

(1)  $\overline{\text{REQ}}$ : 总线请求三态双向信号, 该信号一旦有效即表明驱动它的设备要求使用总线。它是一个点到点的信号线, 任何主设备都有其自身的  $\overline{\text{REQ}}$  信号。

(2)  $\overline{\text{GNT}}$ : 总线请求允许三态双向信号, 用来向申请使用总线的设备表示允许。这也是一个点到点的信号线, 任何主设备都应有自己的  $\overline{\text{GNT}}$  信号。

#### 5. 错误报告信号

为使数据传输可靠、完整, PCI 总线标准要求所有挂于 PCI 总线上的设备都应该具有错误报告线。

(1)  $\overline{\text{PERR}}$ : 数据奇偶校验错误报告信号, 在数据传送过程中, 该信号有效表示出现一次奇偶校验错误。

(2)  $\overline{\text{SERR}}$ : 系统错误报告信号, 该信号漏极开路, 以逻辑的形式允许多个设备共同驱动和分享。该信号的作用是报告地址奇偶错、特殊命令序列中的数据奇偶错, 以及其他可能引起灾难性后果的系统错误。

#### 6. 中断接口信号

PCI 总线中共有四条中断请求线, 分别是  $\overline{\text{INTA}}$ 、 $\overline{\text{INTB}}$ 、 $\overline{\text{INTC}}$  和  $\overline{\text{INTD}}$ 。中断信号在 PCI 总线中为可选项, 中断信号低电平有效, 使用漏极开路方式驱动。此类信号的建立与撤销与时钟不同步。对于单功能设备, 只有一条中断线, 而多功能设备最多可有四条中断线。如果一个设备要实现一个中断, 就定义为  $\overline{\text{INTA}}$ ; 要实现两个中断, 就定义为  $\overline{\text{INTA}}$  和

$\overline{\text{INTB}}$ 。依此类推。所谓多功能是指，将几个相互独立的功能集中在一个设备中。

### 7. 高速缓存支持信号

(1)  $\overline{\text{SBO}}$ ：探测返回双向信号，该信号有效，表示命中了一个修改行，以支持写贯穿或回写操作。

(2)  $\text{SDONE}$ ：查询完成双向信号，用来表示当前查询的状态。该信号无效（低电平），表示查询仍在进行，否则，表明查询已经完成。

### 8. 系统测试信号

PCI 总线提供 5 个系统测试信号，它们是  $\text{TCK}$  测试时钟、 $\text{TDI}$  测试数据输入、 $\text{TDO}$  测试数据输出、 $\text{TMS}$  测试模式选择和  $\overline{\text{TRST}}$  测试复位。

必须注意的是，如果 PCI 进行 32 位操作时，上述信号中  $\text{CLK}$ 、 $\overline{\text{RST}}$ 、 $\text{AD31} \sim \text{AD0}$ 、 $\text{C}/\overline{\text{BE}} 3 \sim \text{C}/\overline{\text{BE}} 0$ 、 $\overline{\text{PAR}}$ 、 $\overline{\text{FRAME}}$ 、 $\overline{\text{IRDY}}$ 、 $\overline{\text{TRDY}}$ 、 $\overline{\text{STOP}}$ 、 $\overline{\text{DEVSEL}}$ 、 $\overline{\text{IDSEL}}$ 、 $\overline{\text{PERR}}$ 、 $\overline{\text{SERR}}$ 、 $\overline{\text{REQ}}$ 、 $\overline{\text{GNT}}$  是必要信号，而其他信号则是任选的。如果要进行 64 位扩展，则以下信号都被使用。

### 9. 64 位总线扩展信号

(1)  $\overline{\text{REQ64}}$ ：64 位传输请求信号，该信号由当前主设备驱动，表示本设备要求采用 64 位通道传输数据，与  $\overline{\text{FRAME}}$  时序相同。

(2)  $\overline{\text{ACK64}}$ ：64 位传输允许信号，表明从设备将启用 64 位通道传输数据，由从设备驱动，与  $\overline{\text{DEVSEL}}$  有相同的时序。

(3)  $\text{AD63} \sim \text{AD32}$ ：扩展的 32 位地址和数据多路复用三态双向线。64 位数据总线和地址总线的高端部分。

(4)  $\text{C}/\overline{\text{BE}} 7 \sim \text{C}/\overline{\text{BE}} 4$ ：总线命令和字节允许多路复用扩展三态双向信号线。在地址传送节拍期内，如果使用了  $\text{DAC}$  命令且  $\overline{\text{REQ64}}$  信号有效，则表明  $\text{C}/\overline{\text{BE}} 7 \sim \text{C}/\overline{\text{BE}} 4$  上传输的是总线命令，否则这些位是保留的且不确定。在数据传送节拍，若  $\overline{\text{REQ64}}$  和  $\overline{\text{ACK64}}$  同时有效时，该 4 条线上传输信息对应说明数据线上哪些字节有意义，如  $\text{C}/\overline{\text{BE}} 4$  对应第 4 字节， $\text{C}/\overline{\text{BE}} 5$  对应第 5 字节。

(5)  $\overline{\text{PAR64}}$ ：奇偶校验三态双向信号，是  $\text{AD63} \sim \text{AD32}$  和  $\text{C}/\overline{\text{BE}} 7 \sim \text{C}/\overline{\text{BE}} 4$  的校验位。该信号与  $\text{AD63} \sim \text{AD32}$  的时序相同，但滞后一个时钟周期。写操作时， $\overline{\text{PAR64}}$  由主设备发出；读操作时， $\overline{\text{PAR64}}$  由从设备发出。

## 4.5 STD 总线

STD 总线（Standard bus）是美国 PROLOG 公司于 1978 年宣布的一种工业标准微机总线，它是一种 56 线的小底板总线。实践证明，STD 总线是最可靠的工业标准总线。

### 1. STD 总线的特点

#### (1) 高可靠性

STD 总线模板上的元器件都经过严格的检验和测试。因此，PROLOG 公司的 STD 总线产品平均无故障间隔可达 60 年。为了适应工业控制的恶劣环境，对该产品的印制板布线、

电源的抗干扰性能、旁路和端点技术、功能划分、各种良好的接地和屏蔽,以及存储器的掉电保护等都采取了相应的措施。此外,为了适应工业控制现场的震动、灰尘、高温、有害气体和各种电磁干扰,采用了固化操作系统、固化系统软件和应用软件,从而使 STD 产品具有在恶劣环境下的生存能力。

### (2) 小板结构, 开放式组态

STD 总线所有模块的标准尺寸为  $165.5 \times 114.3 \text{mm}^2$ 。这种小板结构加上牢固机架有较好的结构强度,具有抗震动、抗断裂、抗变应力、抗老化、抗干扰等优点。小板上元器件少,产生的热量也少,一般的风扇或空气对流即可冷却,使之适应工业现场的环境。另外,由于元器件少,也便于诊断和排除故障,从而提高了系统的可靠性和可维修性。

STD 总线提供了开放式的结构,系统的组成没有固定的模式和标准机型,可由近千种模块,根据需要像“搭积木”一样拼装出自己的控制系统。

STD 模块设计非常标准,信号流向基本上都是由总线驱动,到功能模块,到 I/O 驱动输出,这种结构设计使各种信号流和数据流尽可能具有最短途径,可提高处理速度,减小分布参量的干扰。另外,由于总线端与 I/O 端放在模板的两端,防止了总线信号与 I/O 信号的相互干扰。

### (3) 兼容式的总线结构

STD 总线采用了兼容开放式结构。该总线支持 Intel 公司的 80/85 系列、Motorola 公司的 68 系列、Zilog 公司的 Z80 系列和美国国家半导体公司的 NSC800 系列。还可兼容 8088、8086、68000 等 16 位 CPU。对系统升级换代或更换 CPU 种类提供了方便,减少了投资。

### (4) 产品配套、功能齐全

STD 总线产品拥有各种工业控制所需的功能模板。如键盘接口高分辨率图形板、光电隔离脉冲计数板、多路 A/D 转换板、光隔离量输入/输出板、串行通信板、调制/解调器板等。可与现场的各种机动设备直接连接,如驱动步进电机、交直流电机等。

## 2. STD总线规范

STD 总线连接器和引出脚可装在一块母板上,该母板允许任何一种模板插在其任一插槽上,构成不同的工业控制机。STD 的母板用来沟通各模板的数据线、地址线等信息,母板会产生时间延迟和地线环路,也容易造成线间阻抗不匹配,增大并行信号的串扰噪声。为此,STD 总线设计了一种高性能母板,一般采用 4 层印制板结构,将电源线和地线做在中间两层,使原来信号线的特性阻抗降为  $60 \Omega$ ,接近总线驱动阻抗。

STD 总线接口缓冲模块靠近 STD 总母板,I/O 接口靠近用户连接器,功能模块在中间。对于没有用户连接器的模板,如 CPU 板、存储器板等,I/O 模块这部分也用做功能模块。

STD 总线规范对模板的尺寸、总线连接器和引脚分配、信号定义和电气标准等都做了规定。下面仅介绍 STD 总线引脚的分配和电气标准。

### (1) 总线引脚分配

STD 总线一共有 56 根,可分为 5 个功能组:

逻辑电源线          6 根,引脚 1~6;

数据总线	8 根, 引脚 7~14;
地址总线	16 根, 引脚 15~30;
控制总线	22 根, 引脚 31~52;
辅助电源线	4 根, 引脚 53~56。

主要引脚说明:

电源线 (引脚 1~6 和 53~56)。这种电源线分别为模拟电路和数字电路提供电源, 第 1、2 脚接+5V, 第 3、4 脚接+5V 的地。第 5、6 脚为负电源线, 为有些动态 RAM 而使用。当第 5 脚作为后备电池供电时, 需要有切断能力, 以免发生冲突, 而第 6 脚可作为直流电源掉电信号, 用以指示  $V_{CC}$  低于规定的操作极限。

数据总线 ( $D_7 \sim D_0$ ): 8 位、双向三态, 当不使用  $D_7 \sim D_0$  时, 所有功能板应呈高阻状态。数据总线可作为扩展地址线 ( $A_{23} \sim A_{16}$ )。

地址总线 ( $A_{15} \sim A_0$ ): 16 位、三态, 用于存储器和 I/O 接口的寻址译码, 由存储器请求和 I/O 请求区分两种操作。高 8 位地址线可作为扩展的数据总线 ( $D_{15} \sim D_8$ )。

控制总线可分为 5 个部分: 存储器和 I/O 控制; 外设定时; 时钟和复位; 中断和总线控制; 串行优先级联。

## (2) 电气规范

逻辑信号特性: STD 总线被设计为同工业标准 TTL 或高速 CMOS 逻辑电平相兼容。TTL 总线模板的  $V_{OH}$  和  $V_{IH}$  的最小值分别不能低于 2.4V 和 2.0V,  $V_{OL}$  和  $V_{IL}$  的最大值分别不能高于 0.5V 和 0.8V。

CMOS 总线模板的  $V_{OH}$  和  $V_{IH}$  的最小值分别不能低于 3.76V 和 3.85V,  $V_{OL}$  和  $V_{IL}$  的最大值分别不能高于 0.37V 和 0.9V。

总线驱动和负载特性: 在模板上每个总线信号只能有一个负载, 总线驱动必须满足  $I_{OL}=24mA$  (TTL 总线模板) 或  $I_{OL}=6mA$  (CMOS 模板) 电流吸收器的要求。

最大额定电压:  $+V_{CC}=+0.5V$ , 超过此值就会损坏板上的元件。

## 4.6 主要外设总线介绍

系统开发时, 经常要与外部总线打交道, 外部总线常常以专用的接插件形式提供给用户使用, 有时也称 (总线) 接口标准。本节对常用的 USB 总线、IDE 总线、SCSI 总线、IEEE 1394 总线、ATA 总线、AGP 总线、IEEE-488 总线、CAN 总线等进行简要介绍。

### 4.6.1 USB总线

USB (Universal Serial Bus) 通用串行总线是外部设备通用的接口标准, 由 Compaq、HP、Intel、Lucent、Microsoft、NEC 和 Philips 七家公司联合推出。USB 总线是一种连接外部设备的机外总线, 最多可连接 127 个设备, 为微机系统扩充和配置外部设备提供了方便。

USB 1.0 标准颁布于 1996 年, 但直到 1998 年 USB 1.1 标准的颁布和支持 USB 功能的 Windows 98 的推出, USB 才真正得到广泛应用。USB 1.0 和 USB 1.1 标准的最大数据传输速率是 12Mb/s, 2000 年公布了 USB 2.0 标准, 将最大数据传输速率提高了 40 倍, 达到

480Mb/s。

### 1. USB的主要性能特点

(1) 即插即用。即插即用 (Plug and Play) 是一种系统的设置方式。通常, 指设备可以带电插拔, 自动设置。连接外设不必关闭主机电源, USB 可自动识别 USB 上的外围设备, 具有自动配置和重新配置外设的能力。

(2) 接口方便。每个 USB 系统中有一个主机, 采用“级联”方式, USB 总线可连接多个外部设备, USB 最多可连接 127 个设备。每个 USB 设备用一个 USB 插头连接到上一个 USB 设备的 USB 插座上, 而其本身又提供一或多个 USB 插座供下一个或多个 USB 设备连接使用。USB 电缆长度可达 5m, 线缆数目为 4 条 (红、黑、绿、白)。

(3) 传输速度快。USB 支持三种速度模式: 低速 1.5Mb/s, 全速 12Mb/s, 高速 480Mb/s。低速适合交互设备, 如键盘、鼠标、摄像头、游戏设备、虚拟现实等; 全速适合音频设备, 如电话、声频、麦克风等; 高速适合影像设备, 如视频、存储器、图像设备等。多个 USB 设备可能分享同一速度的信道, 显然, 设备越多, 传输速率越低。

### 2. USB的物理接口

USB 总线的电缆有 4 根信号线, 其中, D+ (绿色) 和 D- (白色) 是一对标准尺寸的双绞信号线, Vbus (红色) 和 GND (黑色) 是一对标准尺寸的电源线。D+ 和 D- 每个时刻的信号状态 (电平) 相反。USB 总线采用半双工传输方式。

USB 设备的电源供给有两种方式: 总线供给方式和自给方式 (设备自带电源)。在总线供给方式中, 每根 USB 电缆提供的电源功率是有限的, 主机为直接连到它的 USB 设备提供电源, 电压为 4.75~5.25V, 最大电流值为 500mA, 并且当 USB 设备第一次被主机检测到时, 设备吸入的电流值应小于 100mA。完全依赖电缆供电的 USB 设备称做总线供电设备, 有后备电源的设备称做自我供电设备。

### 3. USB系统组成及拓扑结构

USB 系统包括 USB 主机、USB 设备和 USB 互连。

USB 主机就是一个带有 USB 主控制器的 PC。在任一个 USB 系统中只有一个主机, 到主计算机系统的 USB 接口被称做主控制器。主控制器可采用硬件、固件或软件相结合的方式来实现。根 Hub 集成在主机系统内, 向上与主总线 (如 PCI 总线) 相连, 向下可提供一或多个连接点。

USB 设备分为 Hub (集线器) 设备和 Function (功能) 设备两大类。Hub 提供到 USB 的附加连接点, 为其他 USB 设备提供扩展端口。功能设备就是一种插在 USB Hub 上的外设, 是完成某种具体功能的硬件设备, 如键盘、鼠标等。USB 设备应具有标准的 USB 接口。

USB 互连指的是 USB 设备与主机的连接和通信方式, 它包括总线拓扑结构、内层关系、数据流模型和 USB 调度表。

USB 总线用来连接各 USB 设备和 USB 主机。USB 在物理上连接成一个层叠的星形拓扑结构, Hub 是每个星的中心, 每根线段表示一个点到点的连接, 可以是主机与一个 Hub 或功能设备之间的连接, 也可以是一个 Hub 与另一个 Hub 或功能设备之间的连接。

由于对 Hub 和电缆传输时间的定时限制, USB 的拓扑结构最多只能有 7 层 (包括根



层）。在主机和任一设备之间的通信路径中最多支持 5 个非根 Hub。

#### 4. 总线协议

USB 是一种查询总线，由主控制器启动所有的数据传输。USB 上所连接的外设通过主机基于令牌的调度协议来共享 USB 带宽。

大部分总线事务涉及 3 个包的传输。当主控制器发出一个描述事务类型和方向、USB 设备地址和端点号的 USB 包时，就开始发起一个事务，这个包被称做“令牌包”，它指示总线上要执行什么事务，寻址的 USB 设备及数据传送方向。然后，事务源发送一个数据包，最后，目标一般还要用一个指示传输是否成功的握手包来响应。

#### 5. USB接口工作原理

USB 设备最大的特点就是即插即用。USB 协议在主机启动或是 USB 设备插入系统的时候都要对设备进行配置。所谓配置，就是按照 USB 协议，在 USB 主机与 USB 设备之间进行的一系列“问答”过程。

一个问答过程是通过主机与 USB 设备的端点 0 进行通信来完成的。USB 设备在插入 USB 端点时，主机都通过默认地址 0 与设备的端点 0 进行通信。在这个过程中，主机发出一系列试图得到描述符的标准请求，通过这些请求，主机得到所有感兴趣的设备信息，从而知道了设备的情况以及该如何与设备通信。随后主机通过发出 Set Address 请求为设备设置一个唯一的地址。这样，配置过程就完成了。以后主机就通过为设备设定好的地址与设备通信，而不再使用默认地址 0。

在配置阶段主机了解了设备端点的使用情况，就可以通过这些端点来进行特定传输方式的通信。例如对于 U 盘，采用批量传输方式，并使用特定的 BULK 端点。U 盘是一个标准的 USB 设备，操作系统带有它的驱动，而不需要编写专门的主机驱动程序。但对于非标准设备，同样采用 BULK 传输方式，需要编写专门的主机驱动程序来实现对 USB 设备的操作。因此，在使用该设备之前必须安装设备驱动程序。

### 4.6.2 IDE总线

IDE (Integrated Drive Electronics) 即集成驱动电子装置，是硬盘控制器的接口标准，使用 40 引脚线缆。而 ATA (AT Attachment, AT 嵌入式) 标准用于连接 AT 上的硬盘驱动器和控制器。各种存储外设的接口都叫做 AT Attachment，IDE 接口最早成为存储外设标准，因此叫做 ATA-1。IDE 的 40 根信号线的定义基本上对应于 AT 总线 (ISA 总线的子集) 的信号，可见，对于 AT 而言，IDE 和 ATA 完成同一事务，两个术语经常互换使用。

IDE 和 ATA 的细微区别是，IDE 强调接口设备已经将驱动器和控制器集成在一起，侧重“集成性”。ATA 强调设备接口很容易直接连接到 AT 总线上，侧重“AT 上的附加装置”。IDE 是非官方术语，ATA 是官方术语。IDE 术语覆盖面较宽，ATA 术语覆盖面相对较窄。

IDE 是 40 条信号线的并行总线，16 位数据线，各种读/写等控制信号与接口时序同步，早期使用 40 线扁平电缆，后来使用 80 线扁平电缆（信号线还是 40 条，另外 40 条全为 GND 信号，用于屏蔽），线缆最大长度为 18 英寸 (0.46m)。

IDE 标准由 ANSI 采纳，已成为当今最普遍的硬盘标准。不过，不只是硬盘，它也

允许连接其他的驱动器，如 CD-ROM、磁带机、可移动磁盘等。ATA 至今已有 7 个版本的标准：

(1) ATA-1 (IDE)。首次规定了硬盘的主盘-从盘工作模式 (Master/Slave)。ATA-1 主板上有一个插口，支持一个主设备和一个从设备，每个设备的最大容量为 512MB，支持的 PIO-0 模式传输速率为 3.3MB/s。ATA-1 定义了 PIO-0/1/2 和 DMA-0/1/2/3 等 7 种操作模式。ATA-1 接口的硬盘大小为 5 英寸，而不是现在主流的 3.5 英寸。

(2) ATA-2。ATA-2 是对 ATA-1 的扩展，也称为 EIDE (Enhanced IDE) 或 Fast ATA。它在 ATA-1 的基础上增加了 PIO-3/4 和 Multiword DMA-1/2 等 4 种模式，将硬盘的最高传输速率提高到 16.6MB/s，增加了逻辑块寻址 (LBA, Logical Block Address) 地址转换方式，突破了 CHS (柱面 Cylinder、磁头 Head、扇区 Sector) 固有的 512MB 的限制，可以支持高达 8.1GB 的硬盘。

(3) ATA-3 (FastATA-2)。ATA-3 没有引入更高速度的传输模式，在传输速率上并没有任何的提升，最高传输速率仍为 16.6MB/s，只是在电源管理方案方面进行了修改，引入了简单的密码保护的安全方案，引入了 S.M.A.R.T 技术 (Self-Monitoring Analysis and Reporting Technology, 自监测、分析和报告技术)。这项技术对包括磁头、盘片、电机、电路等硬盘部件进行检测，通过检测电路和主机上的监测软件对被监测对象进行检测，将其运行状况和历史记录同预设的安全值进行分析、比较，当超出安全值的范围时，会自动向用户发出警告，进而对硬盘潜在故障做出有效预测，提高了数据存储的安全性。

(4) ATA-4。ATA-4 定义了 Ultra DMA-0/1/2 等 3 种数据传输模式，Ultra DMA 采用的是 Double Data Rate (双倍数据传输) 技术，让接口在一个时钟周期内传输数据两次，时钟上升和下降期各有一次数据传输，这样数据传输速率一下从 16MB/s 提升至 33MB/s。因此也习惯称 ATA-4 为 Ultra DMA 33 标准或 ATA33 标准。ATA-4 还引入了冗余校验技术 (CRC)，保障了高速传输数据的安全性。ATA1~3 标准只支持硬盘，而 ATA4 及其以后的标准既支持硬盘，也支持光驱。

(5) ATA-5。ATA-5 也称为 Ultra DMA 66 或 ATA 66，它引入了 Ultra DMA-3/4 等 2 种数据传输模式。ATA-5 使主机接收/发送数据速率达到 66.6 MB/s，是 Ultra DMA 33 的两倍，保留了冗余校验技术 (CRC)。为保障数据传输的准确性，防止电磁干扰，ATA-5 开始使用 40 针 80 芯的电缆，40 针脚是为了兼容以往的 ATA 插槽，避免成本的增加。80 芯中新增的都是地线，与原有的信号线一一对应，这种设计可以降低相邻信号线之间的电磁干扰。

(6) ATA-6。ATA-6 使硬盘的外部传输速率达到 100MB/s，故称为 Ultra DMA 100 或 ATA 100，使用 40 针 80 芯的数据传输电缆，向下兼容，支持 ATA33、ATA66 接口的设备完全可以继续在 ATA100 接口中使用。ATA-6 将 LBA 参数由 28 位提升到 48 位，使硬盘的容量突破 128GB 的限制。理论上，支持的硬盘容量可达  $128 \times 2^{20}$  GB。

(7) ATA-7。ATA-7 是 ATA 标准目前最后一个版本，支持 133 MB/s 数据传输速率，也叫 ATA133。

在奔腾 PC 中，ATA 总线的主控制器大多集成在南桥芯片中，通常提供 2 个以上的 ATA/IDE 端口。每个端口连一条 ATA 线缆，可连接 2 个 IDE 驱动器。

### 4.6.3 SCSI总线

SCSI (Small Computer System Interface) 是小型计算机系统接口, 具有应用范围广、多任务、带宽宽、CPU 占用率低, 以及热插拔等优点, 但较高的价格使得它很难像 IDE 硬盘一样普及, 因此 SCSI 硬盘主要应用于中、高端服务器和高档工作站中。

SCSI 是一个高速智能接口。早期 SCSI 接口在小型机上使用, 后多用于工作站、服务器等中高档设备。由于个人计算机在性能、扩充等方面需求的增大, 使 SCSI 在普通 PC 中的应用也越来越多。

#### 1. SCSI的特点

SCSI 具有如下特点:

(1) SCSI 可支持多个设备。SCSI-2 最多可接 7 个 SCSI 设备, SCSI-3 最多可接 15 个 SCSI 设备。也就是说, 所有的设备只需占用一个 IRQ, 同时 SCSI 还支持不同类型的设备, 如 CD-ROM、DVD、数码相机、打印机、硬盘、磁带机、扫描仪等。

(2) SCSI 设备是对等关系。SCSI 设备既可以是启动设备 (发出命令的设备), 也可以是目标设备 (接收命令的设备)。根据操作确定设备性质。

(3) SCSI 允许在对一个设备传输数据的同时, 另一个设备对其进行数据查找。这就可以在多任务操作系统如 Linux、Windows NT 中获得更高的性能。

(4) SCSI 占用 CPU 极低, 在多任务系统中占有优势。由于 SCSI 卡本身带有 CPU, 可处理一切 SCSI 设备的事务, 主机 CPU 只要向 SCSI 卡发出工作指令, SCSI 卡就会工作, 工作结束后返回工作结果给 CPU。

(5) SCSI 有 2 种数据传输方式: 同步方式和异步方式。同步方式是默认方式。

(6) SCSI 设备具有智能化。SCSI 卡自己可对 CPU 指令进行排队, 这样就提高了工作效率。在多任务时硬盘会在当前磁头位置, 将邻近的任务先完成, 再逐一进行处理。

SCSI与IDE/ATA是完全不同的接口, IDE接口是普通PC的标准接口, 而SCSI并不是专门为硬盘设计的接口, 是一种广泛应用于小型机上的高速数据传输技术。在性能方面, SCSI控制器上有一个相当于CPU功能的控制芯片, 能够处理大部分工作, 而IDE整体性能表现一般, CPU占用率明显高于SCSI。在价格方面, SCSI主要针对商业用户专业应用, SCSI产品档次普遍比IDE产品高, 例如转速、缓存、数据传输速率等, 因而价格较高, 而IDE产品价格比较低廉, 主要针对桌面型 电脑应用。在易用性方面, 由于产品的构造原因, SCSI硬盘的使用比较复杂, 而IDE设备仅有主、从设备之分, 在同一数据线上只有两个设备, 技术含量低于SCSI。在设备扩展功能方面, SCSI扩展能力较强, 最多可以连接 15 个设备, 而标准IDE接口最多只能连接 2 个设备, 采用特殊技术的主板最大也只能支持 8 个设备。

#### 2. SCSI的类型

SCSI 有以下几种类型, 比较流行的版本是 SCSI-2。

(1) SCSI-1。SCSI-1 是最早的版本, 现在基本被淘汰, 异步传输的速率为 3MB/s, 同步传输的速率为 5MB/s。

(2) SCSI-2。早期的 SCSI-2, 称为 Fast SCSI, 通过提高同步传输的速率使数据传输速

率从原有的 5MB/s 提高为 10MB/s, 支持 8 位并行数据传输, 可连 7 个外设。后来出现的 Wide SCSI, 支持 16 位并行数据传输, 数据传输速率也提高到了 20MB/s, 可连 15 个外设。使用 50 针或 68 针的接口, 主要用于扫描仪、CD-ROM 驱动器及老式硬盘。

(3) SCSI-3。SCSI-3 又称为 Ultra SCSI, 数据传输速率达到 20MB/s, 若使用 16 位传输的 Wide 模式, 数据传输速率可提高至 40MB/s。使用 68 针的接口, 主要应用在硬盘上。SCSI-3 的典型特点是将总线频率大大提高, 并降低信号的干扰, 以此来增强其稳定性。

### 3. SCSI接口信号

基本 SCSI-2 使用 50 芯线缆, 其中定义了 18 个差分驱动信号线, 其余的线为电源、地、保留, 18 个信号线中, 有 8 个数据线、1 个奇偶校验线、9 个控制/状态线。而 68 芯线缆将数据线扩充到了 16 位。

9 个控制/状态线的含义如下。

**BSY:** 忙状态信号, 可被总线中任何一个启动设备或目标设备置为有效, 表示当前总线忙。

**SEL:** 选择信号, 用于启动设备选择一个目标设备, 或者, 目标设备重新选择一个启动设备, 由 I/O 信号线确定选择方向。

**C/D:** 控制/数据信号, 用于目标设备指示当前数据线上的信息是控制信息 (如命令、状态、消息) 还是数据信息。

**I/O:** 输入/输出信号, 用于目标设备指示当前数据线上信息的传送方向。I/O=1, 输入, 信息送入启动设备。I/O=0, 输出, 信息送至目标设备。

**MSG:** 消息信号, 用于目标设备指示当前数据线上的信息是消息。

**REQ:** 请求信号, 用于目标设备请求一次数据线上的信息传送。命令、数据、状态、消息的每一次传送都要求有 REQ/ACK 的握手信号进行联络。

**ACK:** 应答信号, 用于启动设备指示对 REQ 的认可。

**ATN:** 注意信号, 用于启动设备通知目标设备, 它有一个消息要发送。

**RST:** 复位信号, 要求总线上的所有 SCSI 设备复位。

SCSI 总线的控制信号比较少, 控制逻辑相对简单。SCSI 信号传送有一个特点: 传输过程由启动设备开始, 选择一个目标设备, 但是, 却由目标设备控制传输过程, 而且, 每一次传送, 都由目标设备提出 REQ, 由启动设备做出 ACK 响应。

## 4.6.4 IEEE 1394 总线

IEEE 1394 是一种高性能串行总线标准 (IEEE 1394 high performance serial bus standard), 具有很高的数据传输速率, 十分适合视频影像的传输。作为一种数据传输的开放式技术标准, IEEE 1394 被应用在众多的领域, 其应用大致可分为三个方面, 一是数字录像机、摄录一体机等家电产品, 二是打印机、扫描仪等计算机外设, 三是硬盘、CD-ROM 等微机内部外设。

IEEE 1394 接口技术由苹果公司率先创立, 苹果公司称之为 Firewire, 很多人习惯称之为火线卡。IEEE 协会在 1995 年正式把它接纳为一个新的工业标准, 编号为 1394, 这就是 IEEE 1394 这个名字的由来。不同的公司根据注册的商标, 对 IEEE 1394 接口有不同的叫

法,例如, Sony 称之为 i.Link, Texas Instruments 称之为 Lynx 等。

标准的 1394 接口可以同时传送数字视频信号以及数字音频信号,相对于模拟视频接口,1394 技术在采集和回放过程中没有任何信号的损失,正是由于这个优势,1394 卡更多地被人们当做视频采集卡来使用。目前,800Mb/s IEEE 1394 总线正在逐渐取代 400Mb/s IEEE 1394 总线。

### 1. IEEE 1394 的特点

IEEE 1394 具有如下特点:

(1) 数据传输速率高。IEEE 1394 中规定传输速率为 100Mb/s 到 400Mb/s。IEEE 1394b 中更高的传输速率是 800Mb/s 到 3.2Gb/s。现在 LSI (大规模集成电路) 通常可能达到的物理速率是 200Mb/s, 所以 400Mb/s 几乎可以满足所有的要求。

(2) 实时性好。接口设备对等,速度快,保证了实时性。不分主、从设备,都是主导者和服务者。这样可以不通过计算机而在两台摄像机之间直接传递数据,也可以让多台计算机共享一台摄像机。IEEE 1394 支持同步和异步两种数据传输模式。

(3) 即插即用。无须设定 ID (识别符) 或终端负载,主节点可以动态确定。

(4) 热插拔。系统在全速工作时,IEEE 1394 设备也可以插入或拆除。

(5) 兼容性好。IEEE 1394 总线可适应台式个人机用户的全部 I/O 要求,并可以与 SCSI 并口、RS232 标准串口、Centronics 接口等接口兼容。

(6) 直接提供电源。采用 6 芯电缆,其中 4 根信号线和 2 根电源线,安装简单。设备之间线缆最长 4.5m,从主机到终节点线缆可达 72m。电源为 8~40V 直流,最大耗电 1.5A。

(7) 连接设备多。IEEE1394 接口可以同时连接 63 个不同设备。

IEEE 1394 的不足主要表现在两个方面,一是应用少,现在支持 IEEE 1394 的设备不是太多,只有一些数码相机与 MP3 等使用高带宽的设备使用 IEEE 1394。二是资源占用高,IEEE 1394 总线需要占用大量的资源,所以需要高速度的 CPU。

### 2. USB和IEEE 1394 的比较

USB 和 IEEE 1394 有许多相似之处,也有一些不同。

在计算机领域,IEEE 1394 与 USB 并列,作为计算机的总线和接口占有重要的地位。USB 主要用于 PC,面向低速 PC 外设;IEEE 1394 则不仅用于计算机,还广泛应用于家电产品,尤其是面向中高端的数码应用。IEEE 1394 有比 USB 更高的速度,在除键盘和鼠标外的各种连接中都有可能使用。

IEEE 1394 支持对等传送方式,这点与 USB 总线不同。在对等总线中,任何一个总线上的设备都可以主动地发出请求。而 USB 总线上的设备,则都是等待主机发送请求,然后做出相应的动作。因而 IEEE 1394 设备更加智能化一些,当然也变得复杂一些,成本高一些。这个特性决定了 IEEE 1394 可以脱离桌面主机,对数字化家电而言尤为重要。

IEEE 1394 总线的拓扑结构与 USB 一样,采用树形结构。树形结构就是所有连接在一起的设备不能形成一个环,否则就可能不能正常工作。不过 IEEE 1394b 提出了一个避免环状结构的方法,在设备连接形成一个圆圈时,也能保证正常工作。

IEEE 1394 和 USB 这类串行总线和 PCI 这类并行总线不一样:前者,两个设备之间如

果经过第三个设备,那么数据也必须从第三个设备穿过,就是说第三个设备也要参与传输。后者,两个设备可以直接传输,不用经过第三个设备。其主要原因,还因为 IEEE 1394 是串行总线,而 PCI 是并行总线。

### 4.6.5 AGP总线

AGP (Accelerated Graphics Port) 是加速图形端口,是英特尔开发的局部图形总线技术。

早期的显示卡通过 ISA 总线或者 PCI 总线与主板连接,但是 ISA、PCI 显示卡均不能满足 3D 图形/视频技术的发展要求。PCI 显示卡处理 3D 图形有两个主要缺点;一是 PCI 总线最高数据传输速率仅为 133MB/s,不能满足处理 3D 图形对数据传输速率的要求。二是需要足够多的显存来进行图像运算,这将导致显示卡的成本很高。AGP 接口把显示部分从 PCI 总线上拿掉,使其他设备可以得到更多的带宽,并为显示卡提供高达 1066MB/s (AGP 4X) 的数据传输速率。AGP 以系统内存为帧缓冲,可将纹理数据存储在其中,从而减小了显存的消耗,实现了高速存取,有效地解决了 3D 图形处理的瓶颈问题。

#### 1. AGP的主要特点

(1) AGP是一种局部图形总线。AGP把显示芯片直接同芯片组的 内存控制器电路相连,实现一种“点对点”的连接,一个系统只能有一个AGP,所以,AGP不会取代PCI总线。

(2) 可直接对系统主内存中的图像数据进行操作。AGP 技术有两个核心内容:一是使用 PC 的主内存作为显存的扩展延伸,增加了显存的潜在容量。二是使用更高的总线频率,比如 66MHz、133Hz 等,极大地提高了数据传输速率。

(3) 采用双泵技术。AGP 利用时钟信号的两个边沿(即上升沿和下降沿)来传输数据,相当于使工作频率提高了两倍。

(4) 两种工作方式:DMA 和 DIME。DMA 方式不使用 PC 的主内存作为显存的扩展,只是利用 AGP 总线的高速传输特性进行数据传送。DIME (Direct memory excute) 方式将主内存的空闲部分映射为显存的扩展(但是,并不能取代显示卡的显存,而只是显示卡的显存的一种扩展和补充),使之与显示卡的显存在操作上成为一体。DIME 是 AGP 的核心技术。

(5) AGP 标准分为 AGP1.0 (AGP 1X 和 AGP 2X)、AGP2.0 (AGP 4X)、AGP3.0 (AGP 8X)。

AGP 1X: 使用 32 位传输通道,工作频率 66MHz,数据传输带宽 266MB/s ( $66\text{MHz} \times 32 \div 8 = 266\text{MB/s}$ ),采用上升沿触发技术,工作电压为 3.3V。

AGP 2X: 使用 32 位传输通道,工作频率 133MHz,数据传输带宽 533MB/s,采用上升沿和下降沿触发技术(双泵技术),工作电压为 3.3V。

AGP 4X: 使用 32 位传输通道,工作频率 266MHz,数据传输带宽 1066MB/s,采用双泵技术,工作电压为 1.5V。

AGP 8X: 使用 32 位传输通道,工作频率 533MHz/s,数据传输带宽 2133MB/s,采用双泵技术,工作电压为 0.8V。

## 2. AGP与PCI的比较

AGP 实质上是 PCI 技术标准的扩充。这也是出于简化开发设计的考虑,使其类似于 PCI 总线。AGP 与 PCI 总线不同,其地址线 and 数据线分离 (PCI 是 49 根信号,而 AGP 是 65 根信号)。地址线和数据线分离,可实现“流水线”处理,提高实际数据传输速率。同时,由于没有切换的“开销”,也提高了随机访问主存时的性能。

PCI 总线数据宽度一般为 32 位,以 33MHz/s 的速率运行,这样,它能提供的最大带宽就是  $33\text{MHz} \times 32 \text{ 位} \div 8 = 133\text{MB/s}$ 。尽管 PCI 64/66 规范提供了 64 位的数据宽度和 66MHz 的工作频率,带宽相应达到了 533MB/s,但它面向的是需要极高数据带宽的 I/O 控制器,比如 IEEE 1394 或者千兆位的网卡。AGP 同样是 32 位的数据宽度,但它的工作频率从 66MHz 开始,AGP 1X 规范利用每个时钟周期的下降沿传输数据,能提供 266MB/s 的带宽;而 AGP 2X 同时利用时钟周期的上升沿和下降沿传输数据,可以达到 533MB/s 的带宽;AGP 8X 更是利用单边 4 次触发技术把带宽提高到了 2133MB/s。

实际上,AGP 和 PCI 根本的区别还是在于 AGP 是一个“端口”,而 PCI 则是一条“总线”。AGP 点对点连接芯片组和图形加速卡“终端”,是一条独占的“专线”,而 PCI 则可以连接许多不同种类的终端,可以是显示卡、网卡、声卡等,所有这些不同的终端都必须共享这条 PCI 总线和它的带宽。

### 4.6.6 IEEE 488 总线

在工业生产、科学研究和实验测量中,经常需要同时使用各种各样的仪器进行测量并需要对其测量结果按照公式进行计算和处理。因此,就需要把许多仪器结合在一起形成一个自动化测量计算系统。但各种仪器仪表的不兼容性给组成系统造成了困难。IEEE 488 总线就是为了把原来互不兼容的设备连成测量计算系统的一种接口和总线标准。

IEEE 488 总线由美国 HP 公司最先提出,到 1975 年形成标准。又称 HP HB,还称 GP-IB 或 IEC-IB。它们是国际标准的通用接口总线,是一种异步双向总线,是专门用于连接系统而不是连接部件或模块的,例如计算机与电压表、信号发生器、程控电源等测量仪器以及各种仪表间的信息通信。

#### 1. IEEE 488 的主要特点

IEEE 488 的特点如下。

- (1) 8 位数据宽度。
- (2) 采用位并行、字节串行、三线异步传送技术,允许不同速度的装置工作在同一系统内。
- (3) 连接的设备最多不超过 15 个,电缆最长不超过 20m,若超过时,需附加硬件或调制解调装置。
- (4) 信号最大传输速率为 1MB/s。
- (5) 有 16 条信号线,采用负逻辑的 TTL 电平,低电平  $\leq +0.8\text{V}$  为“1”,高电平  $\geq +2.0\text{V}$  为“0”。
- (6) 地址容量为听地址 31 个,讲地址 31 个,地址可扩展到 961 个。

## 2. IEEE 488 总线结构

从组成部件的功能来看, IEEE 488 总线结构包括控制器、接收器、发送器。被连接的设备可以是发送器, 可以是接收器, 还可以既是发送器又是接收器, 但总线中必须有一个设备同时是控制器、发送器和接收器(通常是指计算机)。在 IEEE 488 总线系统工作时, 任一时刻只能一个发送器工作, 但接收器可以是一个也可以是多个, 至于哪个设备做发送器, 哪个设备做接收器, 由控制器根据工作需要确定。因此, 连在系统中的设备可以完成如下的一种或多种功能:

(1) 控制器。可向 IEEE 488 总线发出控制命令来管理整个系统的通信。例如启动某设备进入受控状态; 指定某发送器和某接收器之间进行通信; 处理某设备的服务请求等。一般要求一个系统中不能有两个以上的控制器。

(2) 接收器。可从控制器获得信息和接收来自其他设备的数据。总线上允许有多个接收器同时工作。

(3) 发送器。作为信息源, 可通过总线发送信息或向其他设备发送数据。总线上不允许有两个发送器同时工作。

总线上的设备都有唯一确定的地址。控制器可以根据实际需要选择一个或多个接收器。

接口总线的结构实际上是由接口和总线两部分构成的。接口, 使系统中的仪器设备能够接收、理解和发送信息, 使系统中的仪器设备之间进行有效的通信; 总线, 是连接各仪器设备接口的, 作为信息传递的通道。

## 3. IEEE 488 总线定义及功能

IEEE 488 总线有 16 条信号线, 分为 3 组, 其中 8 条双向数据总线, 3 条信号交换线, 5 条接口管理控制线。

### (1) 数据总线 $D_7 \sim D_0$

由于 IEEE 488 总线没有完善的地址总线和控制总线, 因此总线要完成传输数据、命令和地址的任务。控制器利用数据总线传送指令、地址, 确定系统中谁是发送器, 谁是接收器, 以及确定对应仪器的有关功能、量程等。发送器利用数据总线把仪器有用信息传给接收器, 且把自身的状态信息传送给控制器。

### (2) 信息交换控制线 (3 条)

DAV (Data Valid): 数据有效线。当 DAV=1 时, 表示数据线  $D_7 \sim D_0$  上的数据是有效的, 接收器可以接收。当 DAV=0 时, 表示数据无效, 接收器不应接收。

NRFD (Not Ready For Data): 未准备好接收数据。NRFD=1, 表示接收器还没有准备好接收 DB 总线上的数据。当 NRFD=0, 表示接收器已准备好接收数据。

NDAC (Not Data Accepted): 数据未接收完毕。NDAC=1, 表示 DB 上的数据没有被接收器所接收完。NDAC=0, 表示数据接收完并准备接收新的数据。

3 条信号线中 DAV 是由发送器操纵的, NRFD 和 NDAC 是由接收器操纵的, 构成了 IEEE 488 总线上数据字节准确传送的联络信号, 称之为三线挂钩技术。

### (3) 接口管理控制线 (5 条)

ATN (Attention): 注意信号。ATN=1, 表示数据线上传送的是接口信息 (地址或命



令)。ATN=0 则表示数据线上传送是数据或设备消息。ATN 也称方式选择线,是由控制器发出的。

**EOI (End or Identify):** 结束或识别线。一般与 ATN 信号联合使用。当 ATN=0, EOI=1 时,表示传送多字节信息数据传送的结束;当 ATN=1, EOI=1 时,控制器则作为识别指令,如控制器收到 SRQ 时,就对系统中的仪器设备进行查询,以识别请求服务的设备。

**SRQ (Service Request):** 服务请求线。它是由接收器或发送器向控制器发出的请求服务信号。SRQ=1,要求控制器中断自己的工作,为发出这一请求的设备服务,对控制器来说,它相当于一根中断线。

**IFC (Interface Clear):** 接口清除线。它由控制器发出,用来初始化系统中的所有设备,其功能与系统复位类似,当 IFC=1 时,使所有设备接口的有源回路在 100ms 内恢复到空闲状态。空闲状态是指设备不占有总线,但其本身仍可工作,既不是发送器,也不是接收器,仅监视着总线。

**REN (Remote Enable):** 远程选通线。它用来设置设备的操作方式。当 REN=1 时,表明让所有仪器设备不受仪器本身面板开关控制,而转向通过仪器接口对仪器的远程控制。远程控制是指仪器设备的功能由外部来控制。当 REN=0 时,设备回到本地控制方式。各设备可以设置解除远程控制的开关,但设备本身不能设置远程状态。当接通电源时,设备自动进入本地状态(用面板开关控制状态)。该信号由控制器发出。

## 4.6.7 CAN总线

现场总线作为当今自动化领域技术发展的热点之一,被誉为自动化领域的计算机局域网,它的出现标志着自动化系统步入一个新时代的开端。

什么是现场总线?现场总线定义为应用在生产现场、在微机化测量控制设备之间实现双向串行数字通信的系统,具有开放式、数字化、多点通信技术等特点,可被广泛应用于制造业、楼宇、交通等领域的自动化系统中。

现场总线技术将专用微处理器置入传统的测量控制仪表,使它们各自都具有数字计算和数字通信能力,成为能独立承担某些控制、通信任务的网络结点。它们分别通过普通双绞线等多种途径进行信息传输联络,把多个测量控制仪表、计算机等作为结点连接成的网络系统,利用公开、规范的通信协议,在位于生产控制现场的多个微机化自控设备之间,在现场仪表与用做监控、管理的远程计算机之间,实现数据传输与信息共享,形成各种适应实际需要的自动控制系统。

CAN 总线是一种现场总线。CAN (Controller Area Network) 通常可译为“控制器局域网”,是由 BOSCH 公司开发的,以多主方式工作,网络上任意一个结点均可以在任意时刻主动地向网络上的其他结点发送信息。当多个结点同时向网上传送信息时只有具有最高优先权的结点可不受影响地继续传输数据。通过 CAN 总线,将传感器、控制器和执行器用串行数据线连接起来。它将电缆按树形结构连接起来,其通信协议相当于 ISO/OSI 参考模型中的数据链路层,网络可根据协议探测和纠正数据传输过程中因电磁干扰而产生的数据错误。

### 1. CAN的主要特点

CAN 特点如下。

- (1) 成本低。
- (2) 总线利用率高。
- (3) 数据传输距离远（长达 1km）。
- (4) 数据传输速率高（高达 1Mb/s）。
- (5) 可根据报文的 ID 决定接收或屏蔽该报文。
- (6) 可靠的错误处理和检错机制。
- (7) 报文不包含源地址或目标地址，仅用标志符来指示功能信息优先级。

## 2. CAN的电气特性

CAN 能够使用多种物理介质进行传输，例如双绞线、光纤等。最常用的是双绞线。CAN 只有两条信号线，称为 CAN\_H 和 CAN\_L，信号使用差分电压传送，静态时均为 2.5V 左右，此时的状态表示为逻辑 1，也可以叫做“隐性”。用 CAN\_H 比 CAN\_L 高表示逻辑 0，称为“显性”，此时，电压值通常为 CAN\_H=3.5V，CAN\_L=1.5V。当“显性”位和“隐性”位同时发送的时候，最后总线数值将为“显性”。这种特性，为 CAN 总线的仲裁奠定了基础。

## 3. CAN的工作原理

当 CAN 总线上的一个结点（站）发送数据时，它以报文形式广播给网络中的所有结点。对每个结点来说，无论数据是否是发给自己的，都对其进行接收。每组报文开头的 11 位字符为标识符，它定义了报文的优先级，这种报文格式称为面向内容的编址方案。在同一系统中标识符是唯一的，不可能有两个站发送具有相同标识符的报文。当几个站同时竞争总线读取数据时，这种配置十分重要。

当一个站要向其他站发送数据时，该站的 CPU 将要发送的数据和自己的标识符传送给本站的 CAN 芯片，并处于准备状态；当它收到总线分配时，转为发送报文状态。CAN 芯片将数据根据协议组织成一定的报文格式发出，这时网上的其他站处于接收状态。每个处于接收状态的站对接收到的报文进行检测，判断这些报文是否是发给自己的，以确定是否接收它。

由于 CAN 总线是一种面向内容的编址方案，因此很容易建立高水准的控制系统并灵活地进行配置。我们可以很容易地在 CAN 总线中加进一些新站而无需在硬件或软件上进行修改。当所提供的新站是纯数据接收设备时，数据传输协议不要求独立的新站有物理目的地址。它允许分布过程同步化，即总线上的控制器需要测量数据时，可由网上获得，而无需每个控制器都有自己独立的传感器。

## 习题

1. 什么是总线？简述微机总线的分类。
2. 什么是总线标准？为什么要制订总线标准？总线标准应包括哪些内容？
3. 在异步控制的总线传送中\_\_\_\_\_。（参考选项：所需时间固定不变，所需时钟周期数一定，所需时间随实际需要可变，时钟周期长度视实际需要而定）
4. 什么是总线主控设备和总线从属设备？
5. 简述 PCI 总线的特点。

6. 简述 PCI 总线中桥接器的作用。
7. 简要说明 PC 总线和 ISA 总线的区别与联系。
8. STD 总线是一种什么总线？它的结构特点是什么？
9. USB 总线有何特点？
10. IDE 总线有何特点？
11. SCSI 总线有何特点？
12. IEEE 1394 总线有何特点？
13. AGP 总线有何特点？
14. IEEE 488 总线有何特点？
15. CAN 总线有何特点？

# 第 5 章

## 存 储 器

### 📖 教学目的和要求

本章介绍半导体存储器中典型 RAM 和 ROM 的主要引脚、基本特性、应用特点，典型 RAM 和 ROM 的应用分析，存储器与微处理器连接的电路设计。要求了解和掌握典型 RAM 和 ROM 的特点，存储器 RAM、EPROM 和 E<sup>2</sup>PROM 的扩展电路的设计。

存储器是计算机的重要组成部分，它能够存储数据和程序，要求它的存储容量要大，速度要快，这样才能满足越来越大的数据处理量和处理速度的要求。通常总是把具有一定容量的、速度较快的存储器作为内存储器；而将存储容量大、速度较慢的存储器，如磁带、磁盘等，作为外存储器。

计算机发展的初期，内存储器的主要材料为磁心，因此体积大、成本高，存取速度也比较慢。随着大规模集成电路技术的发展，目前半导体存储器在计算机中的应用越来越广泛，特别是在微型计算机中，几乎全都采用半导体存储器作为内存。所以，本章只介绍半导体存储器。

## 5.1 半导体存储器概述

### 5.1.1 半导体存储器的分类

半导体存储器从使用功能上分类，可分为：随机存取存储器 RAM（Random Access Memory）和只读存储器 ROM（Read Only Memory）两类。RAM 主要用于临时存入各种现场的输入/输出数据、中间计算结果、与外存交换的数据及堆栈，其存储单元的内容既可以读出，也可以写入或改写。ROM 中的信息在使用时一般是不能改变的，它只能读出，所以一般用来存放固定的程序和各种常数、函数表等。

在 RAM 中又可分为两类，分别是双极型（Bipolar）RAM 和 MOS RAM。

① 双极型 RAM 的特点是存取速度比较快，通常可在几十纳秒（ns）之内，但是集成度较低，功耗也大，目前已逐渐被速度日益提高的 MOS RAM 所取代。

② 用 MOS 器件构成的 RAM，又可分为静态（Static）RAM（SRAM）和动态（Dynamic）RAM（DRAM）两种。静态 RAM 采用由六管构成的触发器作为基本存储单元，所以信息保存稳定，而动态 RAM 采用单管构成基本存储单元，靠电容存储电荷的方法保存信息，由于存在电容泄漏，会使信息丢失，所以必须采用刷新（再生）的方法保护信息，在无数据读出时典型要求是每 2ms 刷新一遍。但是动态 RAM 的集成度高，功耗也低，所以它们分别适用于不同的场合。但不论何种 RAM，一旦电源掉电后，RAM 中的信息就会丢失，所以它们只能暂存某些数据和中间结果。对于需要长期保存及使用的程序及数据：利用外部存储器加以存储；使用 ROM。

ROM 分为三种：①掩膜 ROM。这是由工厂按特定线路制造好的，适用于批量生产，成本较低。②PROM（Programmable ROM）。这是一种可一次性写入内容，之后不能修改的 ROM，适用于用户自行设计一种批量定型产品。③EPROM（Erasable PROM）。这种 ROM 可以写入，也可以擦除，然后再写，可以改写多次。但是由于写的速度比较慢，且需要一些额外条件，故使用时仍作为只读存储器。按擦除的方式又可分为紫外线可擦除 PROM（UV EPROM）、电可擦除 PROM（E<sup>2</sup>PROM）和快速电擦除（Flash Memory）三种。

只读存储器 ROM 的电路比 RAM 简单，集成度更高，成本更低，且掉电后信息不会丢失，所以在计算机中把一些管理、监控程序，操作系统的基本输入/输出程序（BIOS）等放

在 ROM 中。

综上所述，半导体存储器的分类可以用图 5-1 表示。

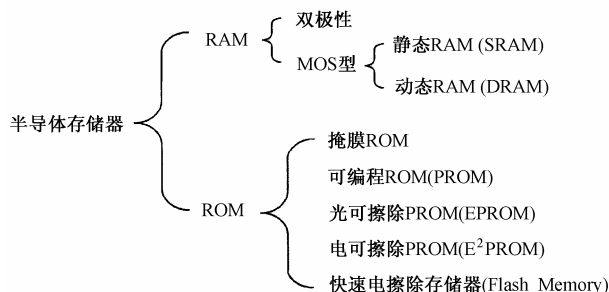


图 5-1 半导体存储器的分类

### 5.1.2 半导体存储器的结构

半导体存储器的基本结构如图 5-2 所示，它由主存储体，地址译码驱动电路、读/写放大电路、时序控制电路四部分组成。

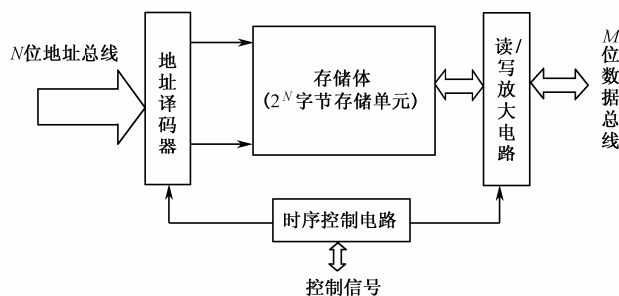


图 5-2 半导体存储器的基本结构

#### 1. 主存储体

存储体是半导体存储器中的核心部分，由大量的存储单元组成。存储单元是存储器中最小的可寻址的单位，CPU 对存储器的访问就是对某个存储单元进行读/写操作。为了实现 CPU 对存储单元的访问，需对存储单元进行顺序编号，该编号被称为地址 (address)。地址与存储单元一一对应，是存储单元的唯一标志。访问存储单元时必须先给出地址，大多数计算机的主存按字节编址，地址码的位数表示 CPU 对存储器进行寻址的空间，如 16 位地址码，寻址空间为 64KB 存储单元。

由图 5-2 可见，存储器地址线的位数为  $N$ ，存储单元数为  $K$ ，它们之间的关系为  $K=2^N$ 。

#### 2. 地址译码驱动电路

地址译码驱动电路用来对地址码进行译码，具有一定的驱动能力，作为地址单元选择线。

### 3. 读/写放大电路

读/写放大电路包括读/写放大器和数据寄存器（三态双向缓冲器），是数据信息的输入/输出通道。

### 4. 时序控制电路

时序控制电路的作用是控制对存储体的访问，以及读/写放大电路中数据信息的流向，时序控制电路所需的控制线包括读（ $\overline{RD}$ ）、写（ $\overline{WR}$ ），或者读/写（ $R/\overline{W}$ ）命令线。

## 5.1.3 半导体存储器的主要性能指标

衡量存储器性能指标主要有三项：容量、速度、价格。

### 1. 存储器容量

通常计算机的编址单元是字节/字（二个字节定义成一个字），存储器的容量是指一个存储器中的存储单元总数，用字或字节数表示。也可以用二进制位（bit）来表示。如 64 千字 =  $64K \times 16$  位， $512KB = 512K \times 8$  位。

除了上述表示容量的单位 B、KB 以外，还采用 MB、GB、TB 等。其中  $1KB = 2^{10}B$ ， $1MB = 2^{20}B$ ， $1GB = 2^{30}B$ ， $1TB = 2^{40}B$ 。

### 2. 存取速度

存储器的存取速度常用存储器的存取时间（Memory Access Time）和存储周期表示，是指访问（读/写）一次存储器所需要的时间。采用 MOS 工艺的存储器，存取周期数为数十至数百纳秒（ns），而双极性型 RAM 的存取周期最快可达 10ns 以下。一般存储周期略大于存取时间，其差别取决于主存的物理实现细节。

### 3. 价格

性能价格比是存储器的重要指标，常以每位价格来描述。若  $S$  位存储器容量的总价格为  $C$ ，则每位价格  $P = C/S$ 。

## 5.2 半导体存储器芯片

了解各种常用半导体存储器芯片的接口特性是用户设计计算机存储器系统的基础。实质上就是了解它与 CPU 总线相关的信号线的功能及工作时序，以便实现存储器芯片上的信号与 CPU 三大总线的连接，构成计算机存储器系统。因此本节分三个层次介绍存储器芯片：首先介绍存储器与 CPU 总线相关的信号线，然后介绍各种常用芯片的外特性及与 CPU 的连接方式。

### 5.2.1 半导体存储器与 CPU 总线相关的信号线

所有存储器与 CPU 总线相关的信号线一般包括三种：地址线（入）、数据线（入/出）、控制线（入），如图 5-3 所示。

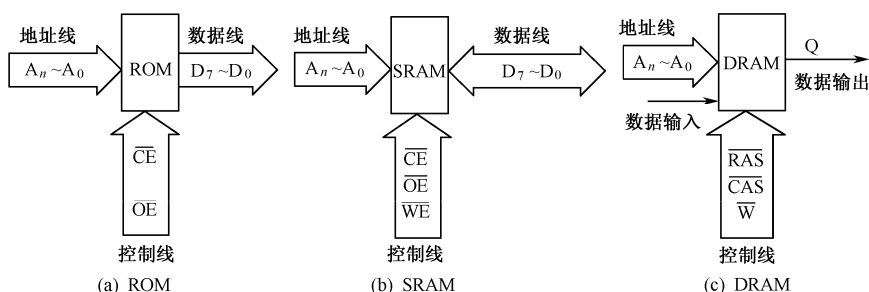


图 5-3 存储器中的信号线

### 1. 地址线 ( $A_n \sim A_0$ )

地址线  $A_n \sim A_0$  用来输入选择存储器中的一个存储单元的地址信号， $A_n$  为最高有效位 (MSB)， $A_0$  为最低有效位 (LSB)，下标  $n$  总比地址引脚数小 1。存储器芯片的存储单元数取决于地址线的位数。对于 1KB 的存储器有 10 条地址线引脚 ( $A_9 \sim A_0$ )，用来选择 1024 个存储单元；具有 11 条地址总线的芯片 ( $A_{10} \sim A_0$ )，有 2048 个存储单元可供使用。对于 8088 CPU，具有 20 位 ( $A_{19} \sim A_0$ ) 地址总线，直接进行选择的存储单元可达 1MB。一般存储器件信息以二进制数 0 或 1 存取。

### 2. 数据线 ( $D_7 \sim D_0$ )

存储器都有一组可以进行输出 (ROM 芯片) 或输入/输出 (RAM 芯片) 的数据总线 ( $D_7 \sim D_0$ )，其中  $D_7$  为最高位 MSB， $D_0$  为最低位 LSB，用于存 (写) / 取 (读) 数据。8 位数据总线意味着一个存储单元存放 8 位 (1 个字节) 数据，当然还有 32 位、16 位、4 位、1 位等数据线的存储器芯片。

### 3. 控制线

#### (1) ROM 控制线

ROM 芯片提供两个控制输入信号：芯片允许  $\overline{CE}$ ，输出允许  $\overline{OE}$ 。

$\overline{CE} = 1$ ：使该芯片处于低功耗备用模式；

$\overline{CE} = 0$ ：该芯片被选中，使  $D_7 \sim D_0$  处于允许状态；

$\overline{OE} = 1$ ：输出被禁止， $D_7 \sim D_0$  处于高阻状态；

$\overline{OE} = 0$ ：允许  $D_7 \sim D_0$  正常输出。

由此可见，要使 ROM 能有效地操作，必须使  $\overline{CE} = \overline{OE} = 0$ 。

#### (2) SRAM 控制线

静态 RAM (SRAM) 芯片提供三个控制输入信号：芯片允许  $\overline{CE}$ ，输出允许  $\overline{OE}$ ，写允许  $\overline{WR}$ 。对 SRAM 进行读或写数据时，必须使  $\overline{CE} = 0$ 。

向 SRAM 写数据时， $\overline{WR} = 0$ ， $\overline{CE} = 0$ ， $\overline{OE} = 1$ ，将  $D_7 \sim D_0$  配置为输入，以实现存储器写操作。

从 SRAM 读数据时， $\overline{WR} = 1$ ， $\overline{CE} = 0$ ， $\overline{OE} = 0$ ，将  $D_7 \sim D_0$  配置为输出，以实现存储器读操作。

注意： $\overline{WR} = \overline{OE} = 0$ ，不能存在； $\overline{WR} = \overline{OE} = 1$ ，数据线处于高阻状态，既不能读也不能写。



（3）DRAM控制信号

动态 RAM（DRAM）提供两个控制信号： $\overline{CE}$ （ $\overline{RAS}$ ， $\overline{CAS}$ ）， $\overline{WR}$ 。

DRAM 和 SRAM 的主要差别在于其容量大，存储器信息依靠电容来保持，因此 2~4ms 之后必须对其刷新一次（即若对 64×1 的 DRAM，每隔 4ms 完成 256 次读操作），需加定时刷新电路。由于容量大，引脚不够，因此采用两路复用锁存方式。用行地址、列地址（ $\overline{RAS}$ ， $\overline{CAS}$ ）选通信号控制两组地址输入，因此 DRAM 接口比较复杂。

DRAM 芯片控制信号没有专门的  $\overline{CE}$  片选线，用  $\overline{RAS}$ 、 $\overline{CAS}$  兼做片选线。只设置一条读/写控制信号  $\overline{WR}$ ， $\overline{WR}=0$  写允许， $\overline{WR}=1$  读允许。

（4）IRAM控制信号

IRAM 是近几年出现的一种新型 DRAM 芯片。该芯片将动态刷新逻辑和地址多路复用逻辑集成于原 DRAM 芯片内部，使之从外部特性看很像 SRAM，其三条控制信号为：芯片允许  $\overline{CE}$ ，输出允许  $\overline{OE}$ ，写允许  $\overline{WR}$ 。

5.2.2 半导体存储器芯片的外特性

所谓存储器芯片的外特性是指与 CPU 总线相关的信号线。

1. ROM芯片外特性

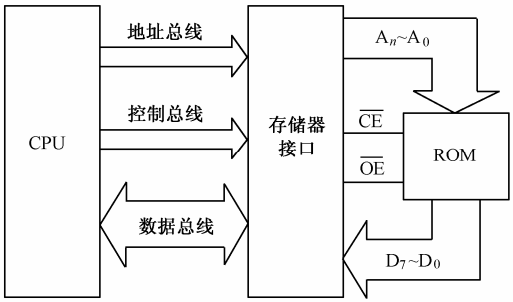


图 5-4 CPU 与只读存储器接口框图

CPU 与只读存储器接口框图如图 5-4 所示。

CPU 从 ROM 中读出一个字节，只要 CPU 提供一个  $A_n \sim A_0$  的地址线， $\overline{CE}=0$ ， $\overline{OE}=0$ ，ROM 数据字节传送到数据总线  $D_7 \sim D_0$ ，CPU 就可以从数据总线上读取数据。

（1）EPROM的外特性

① 常用 EPROM 型号。当前有大量的标准 EPROM 集成电路（IC）器件可供选择使用，但使用比较广泛的是较典型的 Intel 公司的 EPROM 芯片，如表 5-1 所示。

表 5-1 标准的 EPROM 芯片

EPROM	宽度	字长	EPROM	宽度	字长
2716	2K×8b=16Kb	2KB	27512	64K×8b=512Kb	64KB
2732	4K×8b=32Kb	4KB	27010	128K×8b=1Mb	128KB
2764	8K×8b=64Kb	8KB	27020	256K×8b=2Mb	256KB
27128	16K×8b=128Kb	16KB	27040	512K×8b=4Mb	512KB
27256	32K×8b=256Kb	32KB			

其中：27256~27040 在设计中是最流行的，而某些老产品，如 2716，2732，很多制造商已不再生产了，但是在小容量开发时仍然使用。

② 常用 EPROM 引脚。2716、2732 均为 24 脚双列直插式封装，2764~27512 为 28 脚



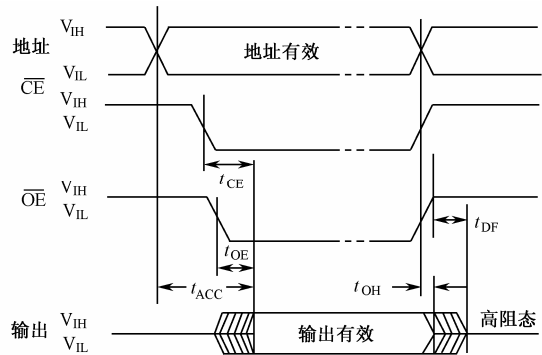


图 5-7 27256 读时序波形图

$t_{ACC}$ 、 $t_{CE}$  的最大值均为 250ns， $t_{DF}$  的最大值为 60ns。

(2) E<sup>2</sup>PROM的外特性

E<sup>2</sup>PROM 是一种在线电可擦除的只读存储器，整个芯片只需+5V 电源  $V_{CC}$  供电。目前用的最多的仍然是 Intel 公司的产品，如 2816, 2817, 2864 等。

① 常用 E<sup>2</sup>PROM 的引脚。2816 和 2817 均为 2KB 容量，每个字节可以在线循环擦写 1000 次，每次写入的数据可以支持 10 年以上。读取数据的速度为 200ns~400ns。2816 和 2817 的引脚排列如图 5-8 所示，其引脚功能见图 5-9。

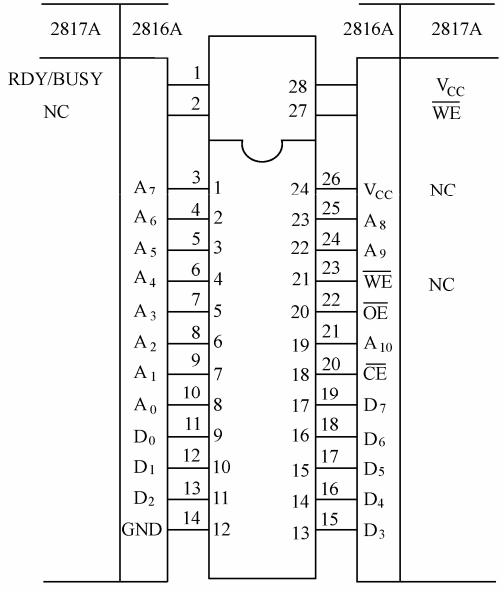


图 5-8 2816 和 2817 引脚排列

芯片引脚	2816A	2817A
$A_0 \sim A_{10}$	地址线	地址线
$\overline{OE}$	输出允许	输出允许
$D_7 \sim D_0$	数据输入/输出	数据输入/输出
$\overline{CE}$	片选	片选
$\overline{WE}$	写允许	写允许
$\overline{RDY}/\overline{BUSY}$	NC	忙/闲输出

图 5-9 2816 和 2817 引脚功能

② E<sup>2</sup>PROM 2817 的工作方式。2817 的工作方式如表 5-2 所示。

③ E<sup>2</sup>PROM 与 CPU 的连接方法。E<sup>2</sup>PROM 与 CPU 的连接方法如图 5-10 所示。8088 提供的 20 位地址线中， $A_{10} \sim A_0$  与 2817 直接连接， $A_{19} \sim A_{11}$  经过译码器译码，译出片选信号  $\overline{CE}$  来选中 E<sup>2</sup>PROM 芯片；8088 提供的 8 位数据线也直接与 2817 连接，8088 提供的  $\overline{WR}$ 、

$\overline{\text{RD}}$ 、 $\text{IO}/\overline{\text{M}}$  经过组合，分别与 2817 的  $\overline{\text{WE}}$ 、 $\overline{\text{OE}}$  连接；2817 的  $\text{RDY}/\overline{\text{BUSY}}$  接计算机的中断请求信号，利用中断方式对 2817 写入。

表 5-2 2817 的工作方式

	$\overline{\text{CE}}$	$\overline{\text{OE}}$	$\overline{\text{WR}}$	$\text{RDY}/\overline{\text{BUSY}}$	$\text{D}_i$
读	0	0	1	高阻	$\text{D}_{\text{OUT}}$
维持	1	×	×	高阻	高阻
字节写入	0	1	0	$\text{D}_{\text{OL}}$	$\text{D}_{\text{IN}}$
整片擦除	字节写入前自动擦除				

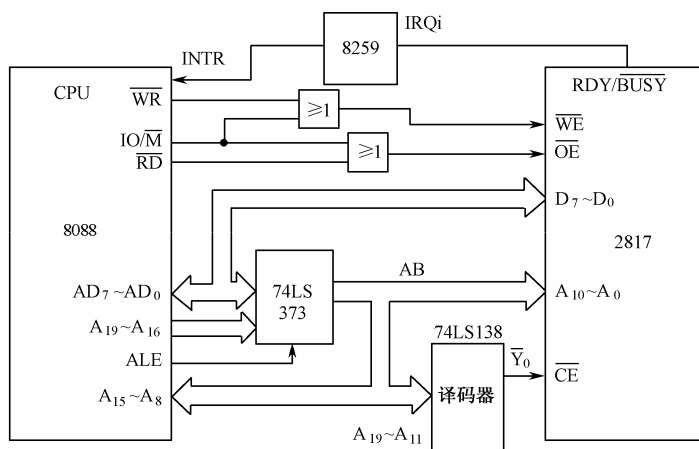


图 5-10 2817 与 CPU 的连接图

## 2. RAM 的外特性

目前广泛使用的 RAM 有两种：SRAM 和 DRAM。其中 SRAM 中数据一旦写入，只要不关电源，数据一直保持有效；而 DRAM 为了保持其中的数据，必须定时对存储器刷新，电路比较复杂，在此不进行介绍。

### (1) 常用 SRAM 型号与引脚

当前可以使用的 SRAM 类型很多，它们之间的主要区别在于密度和组成结构不同。常用的标准 SRAM 芯片有：6216（2KB）、6232（4KB）、6264（8KB）、62128（16KB）、62256（32KB）等。SRAM 的外部引脚排列如图 5-11 所示。

由图 5-11 可见，SRAM 外部引脚有三条控制线，分别为  $\overline{\text{OE}}$ （读允许线）、 $\overline{\text{WE}}$ （写允许线）和  $\overline{\text{CE}}$ （芯片有效线）。

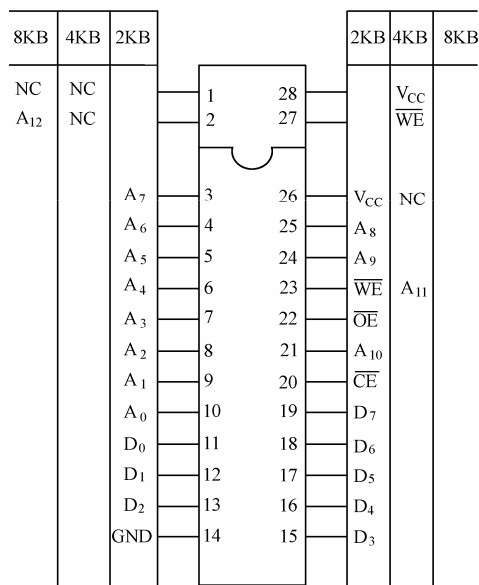


图 5-11 2KB、4KB、8KB SRAM 引脚排列

## （2）SRAM读/写时序

SRAM 读/写时序如图 5-12 所示。

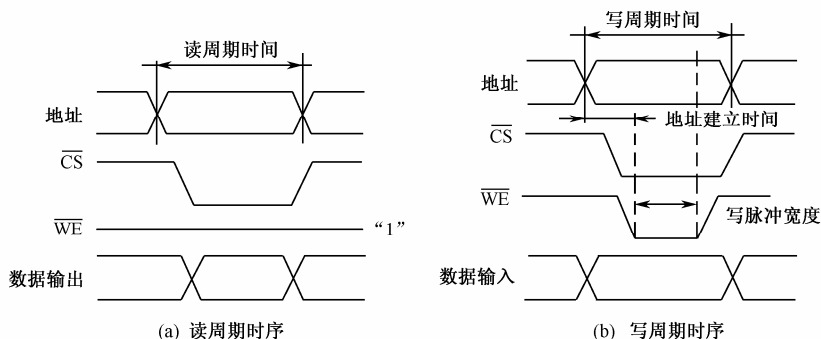


图 5-12 SRAM 读/写时序

## （3）SRAM与CPU的连接

低位地址线、数据线、电源线直接与 CPU 相连接；高位地址线经过译码器译码后，与 SRAM 的  $\overline{CE}$  连接；控制总线组合形成写/读控制  $\overline{WE}$  或  $\overline{OE}$ 。SRAM 与 CPU 的连接方法如图 5-13 所示。

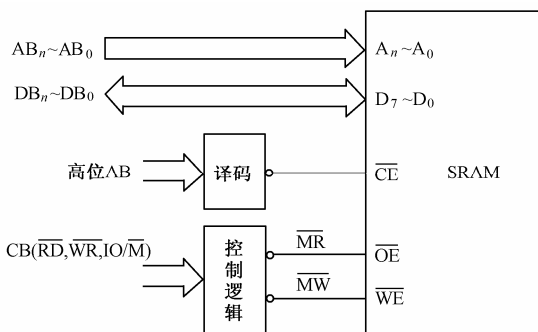


图 5-13 SRAM 与 CPU 的连接方法

# 5.3 半导体存储器的应用

目前，CPU 的种类比较繁多，而不同种类的 CPU 和存储器的连接方法多少有所不同。本节主要介绍半导体存储器在 8088 CPU 系统中的应用。

8088 CPU 有 20 条地址线，可以寻址的物理空间为 1MB，其线性地址范围为 00000H~0FFFFH。有 8 条数据线。用于对存储器操作的主要控制线有  $\overline{IO/M}$ 、 $\overline{RD}$  和  $\overline{WR}$  3 条。其中在对存储器操作时，必须使  $\overline{IO/M}=0$ ， $\overline{IO/M}$  和  $\overline{RD}$  信号经过或门电路形成新的信号，接存储器的  $\overline{OE}$ ； $\overline{IO/M}$  和  $\overline{WR}$  信号经过或门电路形成新的信号，接存储器的  $\overline{WE}$ 。

## 5.3.1 半导体存储器电路的分析方法

所谓半导体存储器电路的分析是指对一个给定的存储器电路，通过分析找出该存储器

电路中各个存储器芯片在该 CPU 系统中所处的地址范围。

通常采用的方法是从各个存储器芯片所需要的控制线、地址线，到 CPU 提供的控制线、地址线逐级写出逻辑函数式，最后得到各个存储器芯片的输入信号和 CPU 提供的信号的逻辑关系式，从而确定各个存储器芯片在该 CPU 系统中所处的地址范围。

【例 5-1】指出图 5-14 中 2764 和 6264 的地址范围，并编程对 6264 清 0。

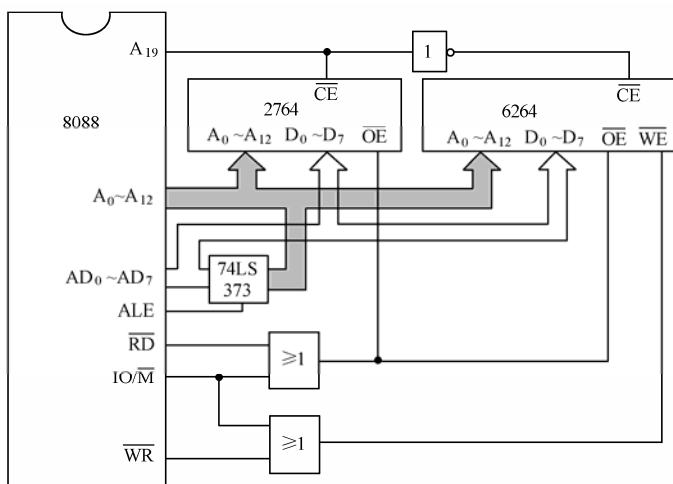


图 5-14 2764 和 6264 利用线选/片选法组成存储器系统

由图 5-14 可知，2764 的  $\overline{\text{CE}}$  接 8088 CPU 的  $A_{19}$ ；2764 的  $A_0 \sim A_{12}$  接 8088 地址总线的  $A_0 \sim A_{12}$ ； $\overline{\text{OE}} = \overline{\text{RD}} \vee \text{IO}/\overline{\text{M}}$ （或运算）。 $\overline{\text{RD}}$  和  $\text{IO}/\overline{\text{M}}$  由 8088 CPU 提供。2764 的  $D_0 \sim D_7$  接数据总线的  $AD_0 \sim AD_7$ 。

8088 CPU 要想对 2764 进行操作，必须满足如下条件：

①  $\overline{\text{OE}} = 0$ ，即  $\overline{\text{RD}}$  和  $\text{IO}/\overline{\text{M}}$  必须同时为 0。也就是说，8088 CPU 必须执行对存储器读操作。例如：MOV AL, [BX] 指令。

②  $\overline{\text{CE}} = 0$ ，即  $A_{19} = 0$ 。

③  $A_{12} \sim A_0$  可以为 0000H~1FFFFH。

从以上条件可以得出，2764 的地址范围为：

	$A_{19}$	$A_{18}$	$A_{17}$	$A_{16}$	$A_{15}$	$A_{14}$	$A_{13}$	$A_{12}$	$A_{11}$	$A_{10}$	$A_9$	$A_8$	$A_7$	$A_6$	$A_5$	$A_4$	$A_3$	$A_2$	$A_1$	$A_0$
首单元地址：	0	×	×	×	×	×	×	0	0	0	0	0	0	0	0	0	0	0	0	0
尾单元地址：	0	×	×	×	×	×	×	1	1	1	1	1	1	1	1	1	1	1	1	1

其中，×可以是低电平，也可以是高电平。

由于  $A_{18} \sim A_{13}$  可以是低电平，也可以是高电平，因此 2764 可以有 64 种不同的地址。也就是说，在存储器应用中利用线选/片选法可以简化硬件电路，但存储器的地址范围不是唯一的，浪费了存储空间。

例如，00000H~01FFFFH，02000H~03FFFFH，04000H~05FFFFH，06000H~07FFFFH 等都是 2764 的地址范围。

同 2764 一样，由图 5-14 可知，8088 CPU 的  $A_{19}$  经过非门接 6264 的  $\overline{CE}$ ； $\overline{OE} = \overline{RD} \vee \overline{IO/\overline{M}}$ （或运算）； $\overline{WE} = \overline{WR} \vee \overline{IO/\overline{M}}$ 。地址线和数据线的连接同 2764。

$\overline{WR}$ 、 $\overline{RD}$  和  $\overline{IO/\overline{M}}$  由 8088 CPU 提供。8088 CPU 要想对 6264 进行操作，必须满足如下条件：

①  $\overline{OE}=0$ ，即  $\overline{RD}$  和  $\overline{IO/\overline{M}}$  必须同时为 0，也就是说，8088 CPU 必须执行对存储器读操作。例如，`MOV AL, [BX]` 指令。或  $\overline{WE}=0$ ，即  $\overline{WR}$  和  $\overline{IO/\overline{M}}$  必须同时为 0，也就是说，8088 CPU 必须执行对存储器写操作。例如，`MOV [BX], AL` 指令。

②  $\overline{CE}=0$ ，即  $A_{19}=1$ 。

③  $A_{12} \sim A_0$  可以从 0000H~1FFFH。

从以上条件可以得出，6264 的地址范围为：

	$A_{19}$	$A_{18}$	$A_{17}$	$A_{16}$	$A_{15}$	$A_{14}$	$A_{13}$	$A_{12}$	$A_{11}$	$A_{10}$	$A_9$	$A_8$	$A_7$	$A_6$	$A_5$	$A_4$	$A_3$	$A_2$	$A_1$	$A_0$
首单元地址：	1	×	×	×	×	×	×	0	0	0	0	0	0	0	0	0	0	0	0	0
尾单元地址：	1	×	×	×	×	×	×	1	1	1	1	1	1	1	1	1	1	1	1	1

其中，×可以是低电平，也可以是高电平。

由于  $A_{18} \sim A_{13}$  可以是低电平，也可以是高电平，因此 6264 可以有 64 种不同的地址。

例如，80000H~81FFFH，82000H~83FFFH，84000H~85FFFH，86000H~87FFFH 等都是 6264 的地址范围。

假设我们选用 6264 的地址范围为 80000H~81FFFH。6264 的单元个数为 8K，即 1FFFH 个。由于 6264 地址范围为 80000H~81FFFH，不妨设其段地址为 8000H，那么 6264 首地址的偏移地址为 0000H，尾地址的偏移地址为 1FFFH。对存储器清 0 编程如下：

<code>MOV AX, 8000H</code>	； 6264 首地址的段地址
<code>MOV DS, AX</code>	； 把段地址存入数据段寄存器
<code>MOV CX, 1FFFH</code>	； 6264 的单元个数
<code>MOV BX, 0000H</code>	； 6264 首地址的偏移地址
<code>MOV AL, 00H</code>	； 对累加器清 0
<code>LOOP1: MOV [BX], AL</code>	； 把 0 存储到存储器单元中
<code>INC BX</code>	； 存储器指针加 1
<code>LOOP LOOP1</code>	； 循环 1FFFH 次

【例 5-2】指出图 5-15 中 2764 和 6264 的地址范围，并编程把 2764 中的内容对应搬运到 6264 中。

由图 5-15 可知，2764 的  $\overline{CE}$  接 74LS138 的  $\overline{Y}_0$ ；2764 的  $A_0 \sim A_{12}$  接地址总线的  $A_0 \sim A_{12}$ ；2764 的  $D_0 \sim D_7$  接数据总线的  $D_0 \sim D_7$ 。 $\overline{OE} = \overline{RD} \vee \overline{IO/\overline{M}}$ （或运算）。6264 的  $\overline{CE}$  接 74LS138 的  $\overline{Y}_7$ ；6264 的  $A_0 \sim A_{12}$  接数据总线的  $A_0 \sim A_{12}$ ；6264 的  $D_0 \sim D_7$  接数据总线的  $D_0 \sim D_7$ 。 $\overline{OE} = \overline{RD} \vee \overline{IO/\overline{M}}$ （或运算）； $\overline{WE} = \overline{WR} \vee \overline{IO/\overline{M}}$ 。 $\overline{WR}$ 、 $\overline{RD}$  和  $\overline{IO/\overline{M}}$  由 8088 CPU 提供。

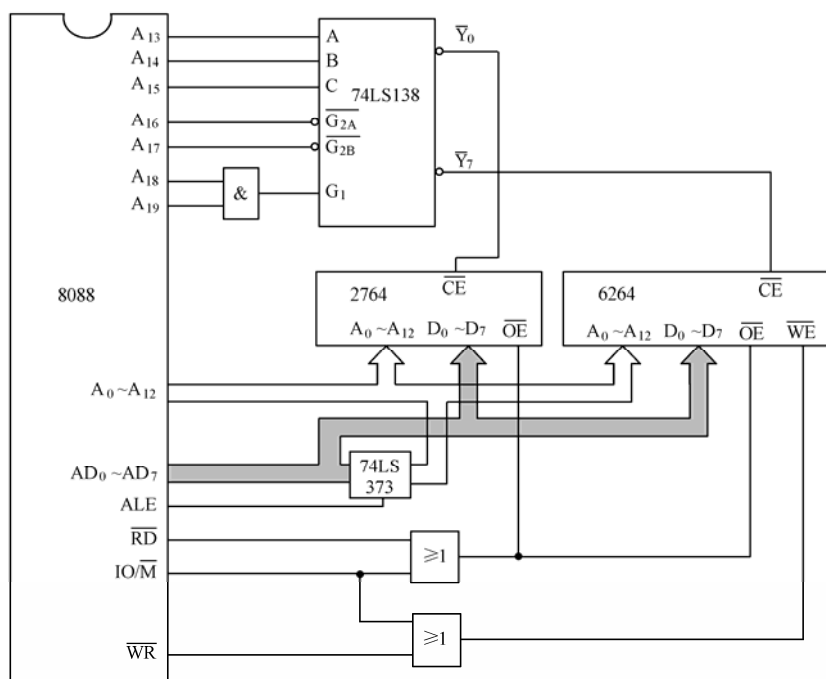


图 5-15 2764 和 6264 利用译码片选法组成存储器系统

8088 CPU 要想对 2764 进行操作, 必须满足如下条件:

①  $\overline{OE}=0$ , 即  $\overline{RD}$  和  $\overline{IO/\overline{M}}$  必须同时为 0, 也就是说, 8088 CPU 必须执行对存储器读操作。例如, `MOV AL, [BX]` 指令。

②  $\overline{CE}=0$ , 即  $\overline{Y_0}=0$ 。

③  $A_{12} \sim A_0$  可以为 0000H~1FFFH。

8088 CPU 要想能够对 6264 进行操作, 必须满足如下条件:

①  $\overline{OE}=0$ , 即  $\overline{RD}$  和  $\overline{IO/\overline{M}}$  必须同时为 0, 也就是说, 8088 CPU 必须执行对存储器读操作。例如, `MOV AL, [BX]` 指令。或  $\overline{WE}=0$ , 即  $\overline{WR}$  和  $\overline{IO/\overline{M}}$  必须同时为 0, 也就是说, 8088 CPU 必须执行对存储器写操作。例如, `MOV [BX], AL` 指令。

②  $\overline{CE}=0$ , 即  $\overline{Y_7}=0$ 。

③  $A_{12} \sim A_0$  可以为 0000H~1FFFH。

本例采用译码片选法。根据 74LS138 的逻辑关系, 可得:  $\overline{Y_0}=0$ , 必须  $A=B=C=0$ ,  $\overline{G_{2A}}=\overline{G_{2B}}=0$ ,  $G_1=1$ ;  $\overline{Y_7}=0$ , 必须  $A=B=C=1$ ,  $\overline{G_{2A}}=\overline{G_{2B}}=0$ ,  $G_1=1$ 。

从以上条件可以得出, 2764 的地址范围为:

$A_{19} A_{18} A_{17} A_{16} A_{15} A_{14} A_{13} A_{12} A_{11} A_{10} A_9 A_8 A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$   
 首单元地址: 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
 尾单元地址: 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1  
 十六进制表示为: C0000H~C1FFFH。

6264 的地址范围为:



A<sub>19</sub> A<sub>18</sub> A<sub>17</sub> A<sub>16</sub> A<sub>15</sub> A<sub>14</sub> A<sub>13</sub> A<sub>12</sub> A<sub>11</sub> A<sub>10</sub> A<sub>9</sub> A<sub>8</sub> A<sub>7</sub> A<sub>6</sub> A<sub>5</sub> A<sub>4</sub> A<sub>3</sub> A<sub>2</sub> A<sub>1</sub> A<sub>0</sub>

首单元地址： 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0

尾单元地址： 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

十六进制表示为：CE000H~CFFFFH。

不妨设段地址为 C000H，那么 2764 首地址的偏移地址为 0000H，尾地址的偏移地址为 1FFFH；6264 首地址的偏移地址为 E000H，尾地址的偏移地址为 FFFFH。把 2764 中的内容对应搬移到 6264 中，编程如下：

```

MOV  AX, C000H
MOV  DS, AX
MOV  SI, 0000H
MOV  DI, E000H
MOV  CX, 1FFFH
LOOP1: MOV  AL, [SI]
        MOV  [DI], AL
        INC  SI
        INC  DI
        LOOP LOOP1

```

### 5.3.2 半导体存储器在计算机系统中的应用

设计方法是指根据给出的实际计算机系统要求，求出实现这一要求的最简单的存储器接口电路。这里所说的“最简单”是指所用的器件数最少，器件的种类最少，器件之间的连线最少，而且所选用的器件资源较为广泛。

设计方法通常按如下步骤进行：

- ① 根据计算机系统要求选择存储器芯片。
- ② 根据存储器芯片和计算机系统情况，确定存储器在计算机系统空间范围。
- ③ 根据存储器地址范围设计存储器接口电路逻辑关系式。
- ④ 选定接口器件的类型。
- ⑤ 根据器件类型变换逻辑表达式。
- ⑥ 根据变换后的接口逻辑表达式设计接口电路。

**【例 5-3】** 为 8088 CPU 的微机系统设计一个 24KB 容量的存储器，要求 EPROM 为 16KB，从 00000H 开始；RAM 为 8KB，从 0C000H 开始。然后编程实现：把两片 EPROM 对应单元中的内容相或，结果存入 RAM 对应的单元中。

为了简化接口设计，我们最好选用相同容量、相同字长的 EPROM 和 SRAM 芯片来组成该存储器。比如，不妨以 2764 作为 EPROM 芯片，以 6264 作为 SRAM 芯片，它们都是 8K×8 位的芯片。这样，根据要求共需要 2 片 2764 和 1 片 6264。关于地址总线接口，可将 20 位地址总线的低 13 位 A<sub>12</sub>~A<sub>0</sub> 直接连到各个存储器芯片的地址线 A<sub>12</sub>~A<sub>0</sub> 上，其余 7 位 A<sub>19</sub>~A<sub>13</sub> 作为高位地址总线，经存储器接口电路提供给 2764 和 6264，作为它们的片选控制信号。8088 CPU 用于对存储器操作的主要控制线有 IO/M、RD 和 WR 3 条。IO/M 和 RD

信号经过或门电路形成新的信号，接存储器的  $\overline{\text{OE}}$ ； $\overline{\text{IO/M}}$  和  $\overline{\text{WR}}$  信号经过或门电路形成新的信号，接存储器的  $\overline{\text{WE}}$ 。

3 片存储器的地址范围：

2764 (1)

$A_{19} A_{18} A_{17} A_{16} A_{15} A_{14} A_{13} A_{12} A_{11} A_{10} A_9 A_8 A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$

首单元地址：0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

尾单元地址：0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1

2764 (2)

$A_{19} A_{18} A_{17} A_{16} A_{15} A_{14} A_{13} A_{12} A_{11} A_{10} A_9 A_8 A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$

首单元地址：0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0

尾单元地址：0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1

6264

$A_{19} A_{18} A_{17} A_{16} A_{15} A_{14} A_{13} A_{12} A_{11} A_{10} A_9 A_8 A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$

首单元地址：0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

尾单元地址：0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1

由以上 3 个存储器芯片的地址范围可以看出：3 个芯片高位地址总线  $A_{19}$ 、 $A_{18}$ 、 $A_{17}$ 、 $A_{16}$  所需要的状态完全一样， $A_{15}$ 、 $A_{14}$ 、 $A_{13}$  分别为 000、001 和 110，所以采用 74LS138 译码器芯片来直接实现它们的片选地址译码是最好不过的了。图 5-16 给出的是采用 74LS138 实现的片选译码电路。

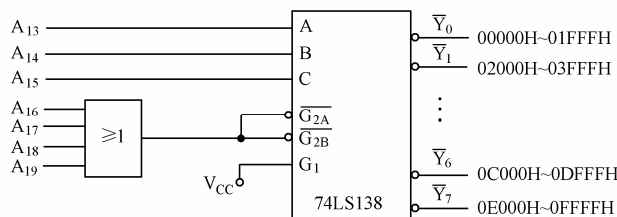


图 5-16 例 5-3 的片选译码电路

8088 CPU 的数据线只要将各存储器芯片上对应的数据线连接在一起即可。8088 CPU 对存储器操作的三条控制线经过或门电路连接存储器各自对应的信号线。最后得到的存储器电路如图 5-17 所示。

不妨设段地址为 0000H，那么 2764 (1) 首地址的偏移地址为 0000H，尾地址的偏移地址为 1FFFH；2764 (2) 首地址的偏移地址为 2000H，尾地址的偏移地址为 3FFFH；6264 首地址的偏移地址为 C000H，尾地址的偏移地址为 DFFFH。把两片 2764 对应单元中的内容相或，结果存入 6264 对应的单元中。编程如下：

```
MOV AX, 0000H      ; 对数据段寄存器赋值
MOV DS, AX
MOV BX, 0000H      ; 2764 (1) 指针
MOV SI, 2000H      ; 2764 (2) 指针
```

```

MOV  DI, C000H      ; 6264 指针
MOV  CX, 1FFFH      ; 循环次数
LOOP1: MOV AL, [BX]  ; 取数
OR   AL, [SI]        ; 运算
MOV  [DI], AL        ; 存数
INC  BX              ; 修改指针
INC  SI              ; 修改指针
INC  DI              ; 修改指针
LOOP LOOP1           ; 循环
    
```

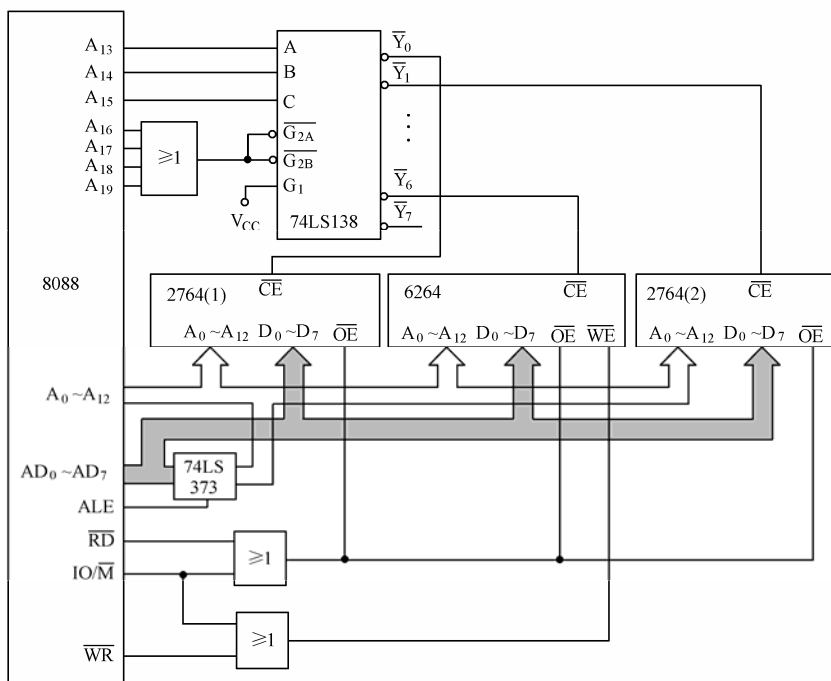


图 5-17 用 2764 和 6264 构成的存储器电路

## 习题

1. 某一个以 8088 为 CPU 的微型计算机系统，内存 RAM 区为 00000H~7FFFFH，若采用 6264、62128、62256，各需要多少片芯片？
2. 利用 2764 芯片，在 8088 系统总线上实现 80000H~8BFFFH 的内存区域，试设计出电路图。
3. 利用全地址译码将 EPROM 2764 接在首地址为 A0000H 的内存区域，试设计出电路图。
4. 简述半导体存储器的种类。
5. 简述各种半导体存储器的区别及各自的用途。

# 第 6 章

## 输入/输出系统

### 📖 教学目的和要求

本章介绍输入/输出系统的一般原理，主要涉及接口概念，CPU 与输入/输出设备之间的接口信号，四种数据传送方式，DMA 控制器的结构、工作原理、编程方法。要求读者重点掌握接口概念、I/O 接口的基础知识、I/O 端口读/写方法和控制技术。

## 6.1 接口概念

输入和输出设备统称为外部设备，它是计算机系统的重要组成部分。程序、原始数据和各种现场采集到的资料和信息，要通过输入装置输入至计算机。计算结果或各种控制信号要输出到各种输出装置，以便显示、打印和实现各种控制。常用的输入设备有键盘、磁卡、操纵杆、鼠标器、光笔、触摸屏、扫描仪等。而处理的结果必须送给要求进行信息处理的人或设备，才能为人或设备所利用，这就必须要有输出设备，如 CRT 显示终端、打印机、绘图仪、记录仪等。

在微型计算机中，CPU 通过接口与各种外部设备交换信息。进行信息的输入/输出时，需要设计把外部设备与微处理器连接起来的电路，即接口电路。由此可见，当实现一个数据的输入/输出操作时，CPU 必须在众多的外部设备中寻找一个确定的设备。如何寻找这一特定的外部设备就是输入/输出寻址所要解决的问题。

CPU 寻址外设可以有两种方式。

### 1. 存储器映射方式

这种方式是将 I/O 端口和存储器单元同等看待，一起编址，相当于给每个 I/O 端口分配一个存储器地址。也就是说，在这种方式中，I/O 端口地址空间就是存储器地址空间的一部分，在存储器的分配图中，预先指定一部分地址空间作为 I/O 空间。这样一来，计算机系统的内存空间中有一部分留做外设地址使用，而剩下的内存空间可为内部存储器使用。

外设与内存统一编址的方法占用了部分内存地址，将外设看做是一些内存单元。因此，原则上说，用于内存的指令都可以用于外设，这给使用者提供了极大的方便。但由于外设占用了内存地址，内存不能再用，因此，就相对地减小了内存的可用范围。而且，从指令上不易区分是寻址内存指令还是用于输入/输出的指令。

### 2. 隔离 I/O 方式

这种编址方式是将 I/O 端口和存储器做不同的处理，分开编址，即两者的地址空间是互相“隔离”的，I/O 结构不会影响存储器的地址空间。例如，在 8086 (8088) CPU 中，内存地址是连续的 1MB，即为 00000H~FFFFFH，而外设的地址范围为 0000H~FFFFH，它们相互独立，互不影响。这是由于 CPU 在寻址内存和外设时，使用了不同的控制信号来加以区分。8088 CPU 的  $\overline{\text{IO}}/\overline{\text{M}}$  信号为 0 时，表示地址总线上有一个内存地址；当它为 1 时，则表示地址总线上的地址是一个有效的外设地址。采用了这种 I/O 编址方式后，处理器对 I/O 端口和存储单元的不同寻址是通过不同的读/写控制信号  $\overline{\text{IO}}/\overline{\text{M}}$ 、 $\overline{\text{RD}}$ 、 $\overline{\text{WR}}$  来实现的。

## 6.2 CPU与I/O设备之间的接口信息

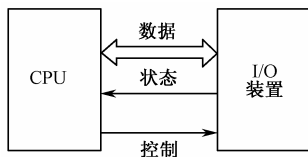


图 6-1 CPU 与 I/O 之间要传送的信息

CPU 与一个外设交换信息，如图 6-1 所示，通常需要有以下一些信号。

### 1. 数据 (Data)

在微型计算机中，数据通常为 8 位、16 位或 32 位的，大致有三种基本类型的数据：

- 数字量信息：一种二进制形式的数据或经过编码的二进制形式的数据。最小单位为“位 (b)”，8 位称为一个字节 (B)。

- 模拟量信息：用模拟电压或模拟电流的幅值大小表示的物理量。
- 开关量信息：只有“开”或“关”两种状态的信息，用一位二进制数表示。

## 2. 状态信息 (Status)

状态信息用来表示外设所处的工作状态。在输入时，有输入装置的信息是否准备好 (Ready)；在输出时，有输出装置是否有空 (Empty)，若输出装置正在输出信息，则以忙 (Busy) 指示。

## 3. 控制信息 (Control)

这类信息是由 CPU 经接口发出的，用于控制外设工作的信号。例如，控制输入/输出装置的启动或停止等。

状态信息和控制信息与数据是不同性质的信息，必须分别传送。因此，外设的状态也必须作为一种数据输入；而 CPU 的控制命令，也必须作为一种数据输出。为了使它们相互之间区分开，它们必须有自己的不同端口地址，如图 6-2 所示。数据需要一个端口；外设的状态需要一个端口，CPU 才能把它读入，了解外设的运行情况；CPU 的控制信号往往也需要一个端口输出，以控制外设的工作。所以，一台外设往往要有几个端口地址，CPU 寻址的是端口，而不是笼统的外设。一个端口的寄存器往往是 8 位的，通常一个外设的数据端口也是 8 位的，而状态与控制端口只用其中的 1 位或 2 位。

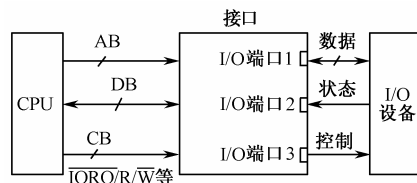


图 6-2 CPU 与外设之间的接口

# 6.3 CPU 与外设之间的数据传送方式

## 6.3.1 无条件传送方式

无条件传送方式是指在传送数据过程中，输入或输出数据一方不查询、判断对方的状态，进行无条件的数据传送。这种传送方式程序设计较为简单，一般用于能够确信外设已经准备就绪的场合。在传送信息时，已知外部设备是准备好的，所以，不查询外设的状态。在输入时，只给出 IN 指令；而在输出时，也只给出 OUT 指令。这种方式必须已知外设已准备好的情况下才能使用，否则就会出错。这样的系统如图 6-3 和图 6-4 所示。

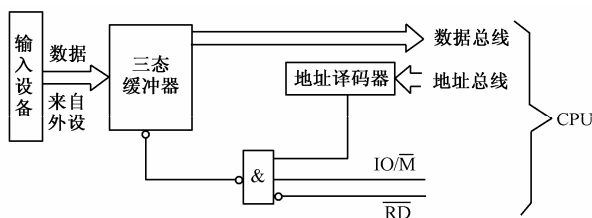


图 6-3 无条件传送的输入方式

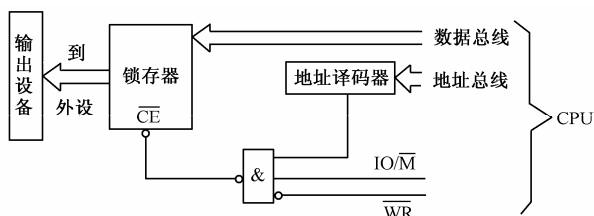


图 6-4 无条件传送的输出方式

### 6.3.2 查询传送方式

查询传送方式就是微型计算机利用程序不断地询问外部设备的状态，根据它们所处的状态来实现数据的输入和输出。

#### 1. 查询式输入

程序控制下的查询式输入方式，在传送前，必须去查询一下外设的状态，当外设准备好了才传送；若未准备好，则CPU就等待。

此方式设计了两个端口，即数据传送端口和状态信号传送端口，其方框图见图 6-5。

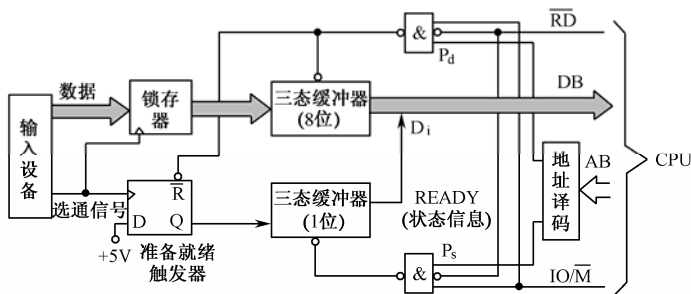


图 6-5 查询式输入的接口电路

当输入设备的数据准备好后就发出一个选通信号，一边把数据送入锁存器，一边使D触发器的Q端电平输出为“1”，给出“准备好”READY的状态信号。当CPU读取状态端口信息时，经过查询状态信息为READY（准备好）后，输入数据，同时状态信息清“0”。

读入的数据是8位的，或16位的，而读入的状态信息往往是1位的，如图 6-6 所示。所以，不同的外设的状态信息，可以使用同一个端口，而只要使用不同的位就行。

查询式输入的程序流程图如图 6-7 所示。

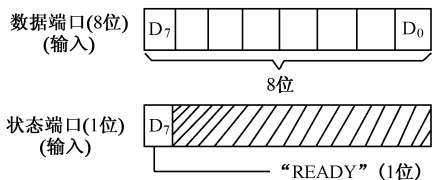


图 6-6 查询式输入时的数据和状态信息

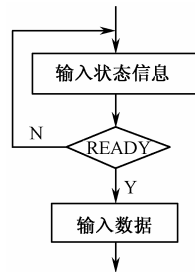


图 6-7 查询式输入的程序流程图

查询部分的程序如下：

```
POLL:  IN  AL, STATUS_PORT    ; 从状态端口输入状态信息
        TEST AL, 80H          ; 检查 READY 是否为 1
        JE  POLL              ; 未 READY, 循环
        IN  AL, DATA_PORT    ; READY, 从数据端口输入数据
```

2. 查询式输出

数据输出过程与输入过程大同小异。CPU 首先通过状态端口读取状态信息，据此判断输出设备是否就绪。若未就绪，“忙”触发器输出仍为“1”，CPU 循环等待、查询；当输出设备准备就绪，即不忙时，CPU 便发出一个  $\overline{\text{ACK}}$  信号，使忙触发器置“0”。CPU 一旦查询得知输出设备不忙，即执行输出指令。地址译码器输出的端口信号  $P_d$  和  $\overline{\text{IO}/\text{M}}$ 、 $\overline{\text{WR}}$  一起，一方面把 CPU 送至数据总线上的数据暂存于锁存器，另一方面使“忙”触发器置“1”，以通知输出设备数据已在锁存器中，并使其进入忙状态。以后每输出一个数据都重复上述过程。如图 6-8 所示。

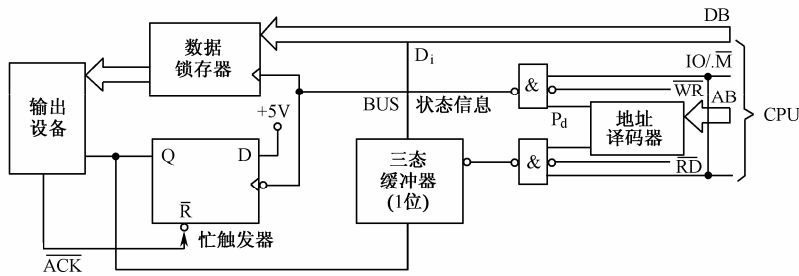


图 6-8 查询式输出接口电路

查询式输出的程序流程图如图 6-9 和图 6-10 所示。

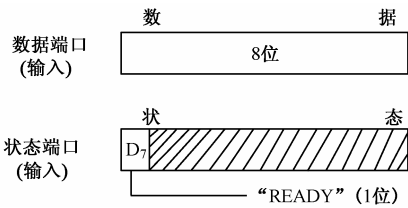


图 6-9 查询式输出的端口信息

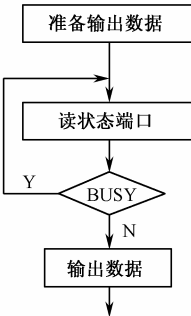


图 6-10 查询式输出的程序流程图

查询部分的程序为：

```
POLL:  IN  AL, STATUS_PORT    ; 从状态端口输入状态信息
        TEST AL, 80H          ; 检查 BUSY 位
        JNE POLL              ; BUSY 则循环等待
        MOV AL, STORE          ; 否则, 从缓冲区取数据
        OUT DATA_PORT, AL    ; 从数据端口输出
```



### 6.3.3 中断传送方式

在中断传送方式中，I/O 设备与 CPU 之间的数据传送是 CPU 通过响应 I/O 设备发出的中断请求来实现的。当 I/O 设备需要 CPU 服务时通过其接口发出中断请求信号；CPU 在收到中断请求后，中断正在执行的程序，执行一个相应的中断服务子程序；服务完毕，返回原来被中断的程序继续执行。

具有这种中断传送方式的输入接口电路如图 6-11 所示。当输入设备准备就绪时，发出就绪状态信号，使数据暂存在锁存器中，同时使中断请求触发器置“1”，向 CPU 发出中断请求信号。如 CPU 响应中断，则执行中断服务程序，由输入指令寻址数据端口并输入数据，同时将中断请求触发器置“0”，以撤销中断请求。CPU 在执行完中断服务程序后自动返回被中断的程序。这样，利用程序中断便完成了输入一个数据的任务。

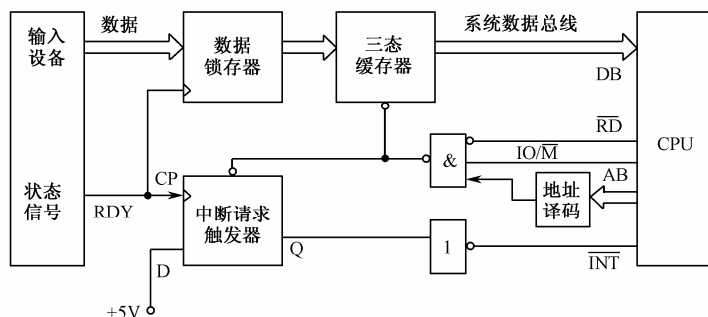


图 6-11 中断传送方式的输入接口电路

### 6.3.4 直接存储器存取（DMA）控制方式

DMA 控制方式下，I/O 设备是和存储器直接交换信息的，不需 CPU 介入。外设与存储器间的数据传输是在硬件的作用下完成的，因此，可使传输速度大大提高。

通常系统的地址、数据总线，以及一些控制信号线（例如  $\overline{\text{IO}/\overline{\text{M}}}$ 、 $\overline{\text{RD}}$ 、 $\overline{\text{WR}}$  等）是由 CPU 管理的。在 DMA 方式时，希望 CPU 把这些总线让出来（即 CPU 连到这些总线上的线处于第三态——高阻状态），由 DMA 控制器接管控制传送的字节数、判断 DMA 是否结束，以及发出 DMA 结束等信号。这些都是由硬件实现的。故 DMA 控制器必须有以下功能：

- ① 能向 CPU 发出 HOLD 信号。
  - ② 当 CPU 发出 HLDA 信号后，接管对总线的控制，进入 DMA 方式。
  - ③ 发出地址信息，能对存储器寻址，能修改地址指针。
  - ④ 能发出读或写等控制信号。
  - ⑤ 能决定传送的字节数，判断 DMA 传送是否结束。
  - ⑥ 发出 DMA 结束信号，使 CPU 恢复正常工作状态。
- 通常 DMA 的工作流程如图 6-12 所示。

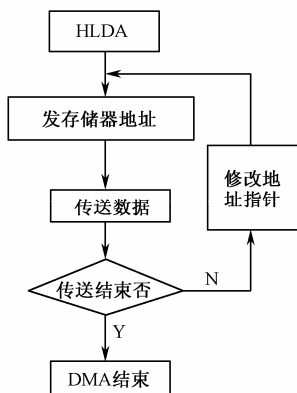


图 6-12 DMA 工作流程图

能实现上述操作的 DMA 控制器的硬件方框图如图 6-13 所示。

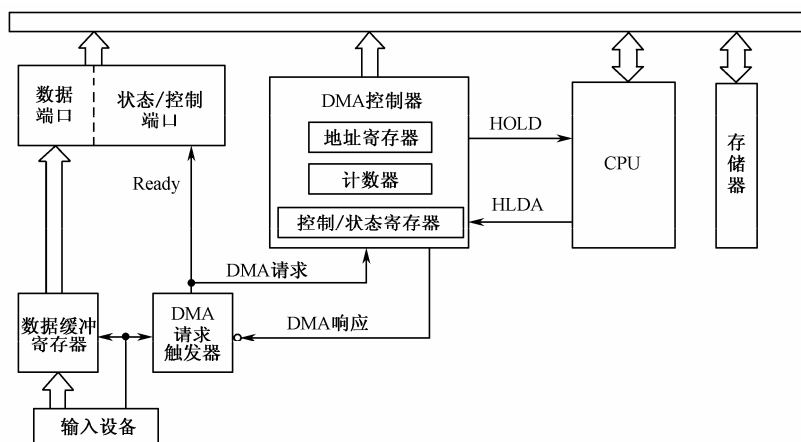


图 6-13 DMA 控制器的硬件方框图

## 6.4 DMA 控制器 8237A

为了高速传送大量数据，微型计算机中采用了直接存储器访问技术，简称 DMA (Direct Memory Access)。一些高速的数据交换，如图像处理、磁盘存取等，适于采用这种方式。

DMA 方式在工作中不用软件指令，而是用硬件控制外设与存储器、存储器与存储器、外设与外设之间的交换数据。在 DMA 工作期间，CPU 不参与工作，DMA 独占总线，此时，CPU 让出地址总线、数据总线和一些控制总线，交由 DMA 控制电路控制，这个控制电路被称为 DMA 控制器。

当进入 DMA 方式时，DMA 控制器就要接管总线。通常，DMA 控制器占用总线有三种方式：CPU 停机方式、周期扩展和周期窃取。

8237A 是一个高性能的 40 个引脚双列直插式可编程 DMA 控制器，其主要功能有：

- ① 在一片内有 4 个独立的 DMA 通道。
- ② 每个通道的 DMA 请求可分别编程允许或禁止。
- ③ 每个通道的 DMA 请求有不同的中断优先级（即 DMA 操作优先权）。优先级有两种：固定优先级和循环优先级，由编程决定。固定优先级的顺序是通道 0 最高，依次是通道 1、通道 2 和通道 3。
- ④ 可在外设与存储器、存储器与存储器之间传送数据，存储器的地址寄存器可以加 1 或减 1。
- ⑤ 可由软件编程改变 DMA 读/写周期长度。
- ⑥ 有四种工作方式：单字节传送方式、数据块传送方式、请求传送方式和级联方式。
- ⑦ 可以多片级联，以扩展通道数。
- ⑧ DMA 操作结束有两种方法：一是字节计数器减 1，由 0 变为 FFFFH；二是外界通过  $\overline{\text{EOP}}$  输入负脉冲，强制 DMA 操作结束。

⑨ DMA 操作启动有两种方法：一是外设输入 DMA 请求信号 DREQ；二是通过软件编程从内部启动。

### 6.4.1 8237A的内部结构

8237A 内部结构框图如图 6-14 所示。8237A 内部有 4 个结构相同的独立通道，图中只画出了 1 个通道的结构，也就是说每个通道都有基地址寄存器（16 位）、基字节数计数器（16 位）、当前地址寄存器（16 位）、当前字节数计数器（16 位）、方式字寄存器（6 位）。由图可知，8237A 内部包括 3 个基本组成部分：控制逻辑块、缓冲器组和寄存器组。

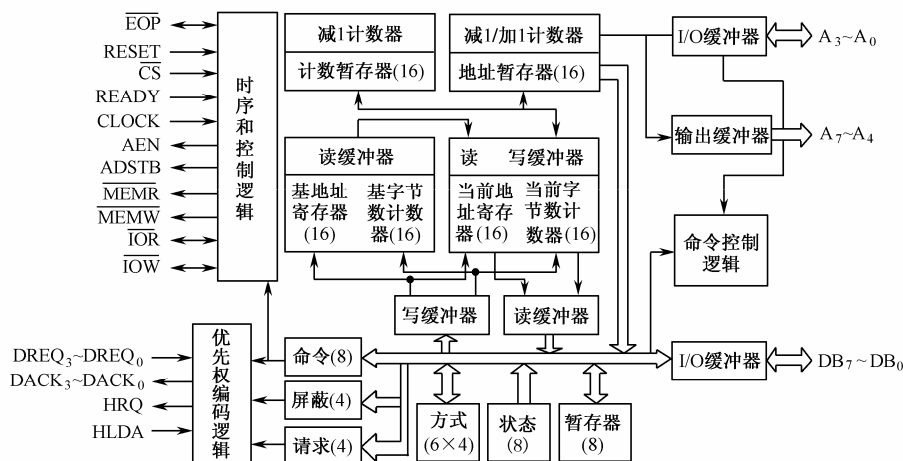


图 6-14 8237A 内部结构框图

#### 1. 控制逻辑块

这组控制逻辑包括三个部分，即时序和控制逻辑、优先权编码逻辑和命令控制逻辑。各部分的逻辑功能如下：

##### (1) 时序和控制

该逻辑模块接收外部时钟及片选和定时读/写信号，用以产生内部的时序控制及对外的控制信号。当 CPU 对 8237A 编程写入或读 8237A 状态时，本逻辑块接收  $\overline{\text{IOR}}$  或  $\overline{\text{IOW}}$  信号，在  $\overline{\text{IOW}}$  有效时，将数据总线的内容写入由地址总线低 4 位 ( $A_3 \sim A_0$ ) 译码所寻址的寄存器；在  $\overline{\text{IOR}}$  有效时，将由低 4 位地址译码所选择的寄存器的内容读到数据总线上。

##### (2) 优先权编码

该逻辑模块对同时提出 DMA 请求的通道进行优先级的排队判优。8237A 有两种优先级编码，即固定优先级和循环优先级编码，可编程选择。在固定优先级编码中，四个 DMA 通道的优先级顺序是固定的，即通道 0 的优先级最高，依次降低的是通道 1、通道 2、通道 3。在循环优先级编码中，最近一次服务的通道被指定为最低优先级。比如原优先级顺序是通道 3 最低，但当通道 1 请求并被服务后，通道 1 就被指定为最低优先级。

##### (3) 命令控制

该逻辑模块在编程时对 CPU 送来的命令字或方式字进行译码，以确定 DMA 服务的类

型。在不同的工作周期内有不同的操作。

## 2. 缓冲器组

缓冲器组包括三个部分，它们分别提供以下地址和数据信号：

①  $A_3 \sim A_0$ ：低 4 位地址线，双向三态信号。CPU 对 DMA 编程时，用于提供对 8237A 内部寄存器寻址的地址信息。在 DMA 操作期间，这 4 位地址线作为寻址存储器的低 4 位地址线。

②  $A_7 \sim A_4$ ：高 4 位地址线，三态输出。这 4 个信号仅用于在 DMA 操作期间提供对存储器寻址的  $A_7 \sim A_4$  的地址信息。

③  $DB_7 \sim DB_0$ ：8 位数据线，双向。在芯片空闲周期（非 DMA 操作周期），用于对片内寄存器写入命令字或方式字，读出状态；在 DMA 操作期间，传送高 8 位地址（ $A_{15} \sim A_8$ ）到系统地址总线，并由 ADSTB 选通锁存到一个 8 位锁存器，与  $A_7 \sim A_0$  组成 16 位地址。在进行存储器与存储器传送期间用于传送读/写数据。

## 3. 寄存器组

8237A 芯片内部共有 12 种寄存器，它们的名称和个数如下：

- |            |        |
|------------|--------|
| ① 基地址寄存器   | 16 位×4 |
| ② 基字节数计数器  | 16 位×4 |
| ③ 当前地址寄存器  | 16 位×4 |
| ④ 当前字节数计数器 | 16 位×4 |
| ⑤ 方式字寄存器   | 6 位×4  |
| ⑥ 暂存地址寄存器  | 16 位×1 |
| ⑦ 暂存字节数计数器 | 16 位×1 |
| ⑧ 状态寄存器    | 8 位×1  |
| ⑨ 命令寄存器    | 8 位×1  |
| ⑩ 暂存寄存器    | 8 位×1  |
| ⑪ 屏蔽寄存器    | 4 位×1  |
| ⑫ 请求寄存器    | 4 位×1  |

可以看出，8237A 内部的寄存器组分成两大类：一类是 4 个通道都有的寄存器，比如基地址和当前地址寄存器、基字节数和当前字节数计数器、方式字寄存器等。另一类是 4 个通道公用的一套寄存器，比如除上述以外的其他寄存器。

### （1）当前地址寄存器

每个通道都有一个 16 位的当前地址寄存器，它保存着 DMA 操作期间的动态地址值，也就是当前的地址值。在每次 DMA 传送一个字节后，这个寄存器的值自动加 1 或减 1（由编程确定）。若编程设为自动初始化，则在每次传送结束后，自动初始化为它的初值（即保存在相应基地址寄存器中的内容）。

这个寄存器可由 CPU 写入或读出，一次操作 8 位。

### （2）当前字节数计数器

每个通道都有一个 16 位的当前字节数计数器，它保存着 DMA 操作期间的尚未传送的字节个数，也就是目前还需要传送的字节数。它的初始值比实际传送的字节数小 1，在每次

DMA 传送一个字节后自动减 1。当该计数器的值由 0 减到 FFFFH 时，产生终止计数信号 TC。在编程为自动初始化情况下，在每次传送结束后，它的值恢复到初始值。

这个寄存器可由 CPU 写入或读出，一次操作 8 位。

### （3）基地址寄存器

每个通道都有一个 16 位的基地址寄存器，它保存当前地址寄存器的初始值。该值是在对 8237A 编程时，与当前地址寄存器同时写入的。该寄存器的作用是在自动初始化情况下，用于恢复相应的当前地址寄存器的初始值。

基地址寄存器一旦写入便不再改变（除非重写），并且不能被 CPU 读出。

### （4）基字节数计数器

每个通道都有一个 16 位的基字节数计数器，它保存当前字节数计数器的初始值。该值是在对 8237A 编程时，与当前字节数计数器同时写入的。该计数器的作用是在自动初始化情况下，用于恢复相应的当前字节数计数器的初始值。

基字节数计数器一旦写入便不再改变（除非重新写入），并且不能被 CPU 读出。

### （5）方式字寄存器

每个通道都有一个 6 位的方式字寄存器，它保存通道的方式控制字，该方式控制字规定了通道的各种操作方式，如工作方式、地址增减选择、传送类型、自动初始化设置等。

方式字寄存器虽然是每个通道一个，但四个通道的方式字寄存器公用一个端口地址。该寄存器只能写入，不能由 CPU 读出。

### （6）暂存地址寄存器和暂存字节数计数器

8237A 内部有一个暂存地址寄存器和一个暂存字节数计数器，它们分别保存当前地址寄存器和当前字节数计数器的内容，用于 8237A 内部过程，用户不能直接使用它们，它们也没有对应的端口地址。

### （7）命令寄存器

这是一个 8 位的寄存器，它保存通道的命令控制字。该命令控制字确定 DMA 请求与响应的有效电平状态、优先级方式、读/写定时方式等。

命令寄存器只能写入，不能读出。但从命令寄存器的端口地址执行读出指令也被认为是合法的，只不过读出的不是命令寄存器的内容，而是状态寄存器的内容。也就是说，命令寄存器与状态寄存器公用一个端口地址，写入的是命令，读出的是状态。

### （8）请求寄存器

这是一个 4 位的寄存器，每一个通道对应 1 位，它保存利用软件编程设置的通道 DMA 请求状态，即软件 DMA 请求。请求寄存器只能写入，不能由 CPU 读出。

### （9）屏蔽寄存器

这是一个 4 位的寄存器，每一个通道对应 1 位，它保存相应通道的对 DMA 请求信号 DREQ 的屏蔽状态，即允许与禁止。屏蔽寄存器只能写入，不能由 CPU 读出。

### （10）状态寄存器

这是一个 8 位的寄存器，它保存每个通道的 DMA 请求状态（DREQ 状态）和 TC 或外部输入 EOP 的状态。

(11) 暂存寄存器

这是一个 8 位的寄存器，故亦称为暂存字节寄存器，它不属于任何一个通道，仅用于在存储器与存储器间传送时暂存传输的数据。

6.4.2 8237A的引脚功能

8237A DMA 控制器有 40 个引脚，双列直插式封装（见图 6-14）。

①  $A_3 \sim A_0$ （Address，输入/输出）：双向三态信号线。在空闲周期，它们是地址输入线，CPU 用这四根地址线选择 8237A 内部不同的寄存器。

②  $A_7 \sim A_4$ （Address，输出）：三态地址输出线，仅在 DMA 操作周期内使用。

③  $DB_7 \sim DB_0$ （Data Bus，输入/输出）：双向三态数据总线，与系统数据总线相连。

④  $\overline{CS}$ （Chip Select，输入）：芯片选择信号，低电平有效。当 8237A 在空闲周期内， $\overline{CS}$  有效时，CPU 可以通过数据线对 8237A 进行读/写操作。可读/写的寄存器端口地址由低 4 位地址线来选择，其对应的端口地址如表 6-1 所示。

表 6-1 8237A 内部端口地址分配

$A_3A_2A_1A_0$	寄存器说明	
0000	通道 0	写：基地址寄存器和当前地址寄存器
	通道 0	读：当前地址寄存器
0001	通道 0	写：基字节计数器和当前字节计数器
	通道 0	读：当前字节计数器
0010	通道 1	写：基地址寄存器和当前地址寄存器
	通道 1	读：当前地址寄存器
0011	通道 1	写：基字节计数器和当前字节计数器
	通道 1	读：当前字节计数器
0100	通道 2	写：基地址寄存器和当前地址寄存器
	通道 2	读：当前地址寄存器
0101	通道 2	写：基字节计数器和当前字节计数器
	通道 2	读：当前字节计数器
0110	通道 3	写：基地址寄存器和当前地址寄存器
	通道 3	读：当前地址寄存器
0111	通道 3	写：基字节计数器和当前字节计数器
	通道 3	读：当前字节计数器
1000	写：命令寄存器；	读：状态寄存器
1001	写：请求寄存器；	读：非法
1010	写：一位屏蔽寄存器；	读：非法
1011	写：方式字寄存器；	读：非法
1100	写：清除 F 触发器；	读：非法
1101	写：8237A 总清；	读：暂存寄存器
1110	写：清除屏蔽寄存器；	读：非法
1111	写：四位屏蔽寄存器；	读：非法

⑤ CLOCK (Clock, 输入): 时钟信号, 5MHz/3MHz。

⑥ RESET (Reset, 输入): 复位信号, 高电平有效。当芯片被复位时, 除了屏蔽寄存器被置位 (4 个通道均禁止 DMA 请求) 外, 其余寄存器均被清 0。

⑦ READY (Ready, 输入): 就绪信号, 高电平有效。当选用慢速存储器或 I/O 设备时, READY 变为低电平。

⑧ DREQ<sub>3</sub>~DREQ<sub>0</sub> (DMA Request, 输入): 外设发出的四个通道的 DMA 服务请求信号, 有效电平由编程决定。在固定优先级情况下, DREQ<sub>0</sub> 优先级最高, 依次减小, DREQ<sub>3</sub> 优先级最低, 但优先级可由编程改变。

⑨ DACK<sub>3</sub>~DACK<sub>0</sub> (DMA Acknowledge, 输出): 8237A 每个通道的 DMA 响应信号, 有效电平由编程决定。

⑩ HRQ (Hold Request, 输出): 总线请求信号, 高电平有效。

⑪ HLDA (Hold Acknowledge, 输入): 总线响应信号, 高电平有效。CPU 在每个总线周期的最后一个 T 状态检查是否有总线请求, 当发现 HRQ 有效后, 就暂停执行程序, 让出总线, 并使 HLDA 有效 (高电平)。

⑫  $\overline{\text{IOR}}$  (I/O Read, 输入/输出): I/O 读, 双向, 三态, 低电平有效。在 8237A 空闲周期, 它是一条输入控制信号, CPU 利用这个信号读取 8237A 内部寄存器状态。在 DMA 传送时, 它是一条输出控制信号, 令外设把数据送至数据线, 由  $\overline{\text{MEMW}}$  信号将数据写入存储器。

⑬  $\overline{\text{IOW}}$  (I/O Write, 输入/输出): I/O 写, 双向, 三态, 低电平有效。在 8237A 空闲周期, 它是一条输入控制信号, CPU 利用这个信号把控制字写入 8237A 内部寄存器。在 DMA 传送时, 这是一条输出控制信号, 利用  $\overline{\text{MEMR}}$  从存储器读出数据, 写入外设。

⑭  $\overline{\text{MEMR}}$  (Memory Read, 输出): 存储器读信号, 三态, 低电平有效。只用于 DMA 传送。

⑮  $\overline{\text{MEMW}}$  (Memory Write, 输出): 存储器写信号, 三态, 低电平有效。只用于 DMA 传送。

⑯ AEN (Address Enable, 输出): 地址允许信号, 高电平有效。它将在锁存器中锁存的高 8 位地址 A<sub>15</sub>~A<sub>8</sub> 送至系统地址总线, 与芯片输出的低 8 位地址 A<sub>7</sub>~A<sub>0</sub> 组成 16 位地址, 也就是说, 在 AEN=1 期间, 芯片输出的 16 位地址有效。

⑰ ADSTB (Address Strobe, 输出): 地址选通信号, 高电平有效。在 DMA 操作时, 它将芯片内数据缓冲器送出的高 8 位地址通过 DB<sub>7</sub>~DB<sub>0</sub> 送到外部地址锁存器中锁存。

⑱  $\overline{\text{EOP}}$  (End of Process, 输入/输出): 过程结束信号, 双向, 低电平有效。作为输入, 如果外部输入一个低电平信号, 将强迫 DMA 传送结束; 作为输出, 当字节数减 1 产生 TC 时 (即减 1 由 0 变为 FFFFH), 在  $\overline{\text{EOP}}$  上输出一个有效的负脉冲, 表示 DMA 传送结束。

### 6.4.3 8237A的工作方式

本节从 DMA 操作过程的角度, 介绍 DMA 控制器 8237A 的几种工作方式, 包括主从模

态、传送方式、传送类型、优先级编码、工作周期、读/写时序、自动初始化方式、存储器到存储器的传送等。

### 1. 主从模态

DMA 控制器既可以作为 I/O 端口接受 CPU 的读/写操作，也可以代替 CPU 占有总线，控制外设与存储器之间传送数据，它充分体现了 DMA 控制器的两大特性，即总线的主控性和总线的从属性。按照这两大特性，DMA 有两种工作模态：主态方式和从态方式。

① 主态方式。在主态方式时，DMA 控制器是总线的控制者，此时，它如同 CPU 一样，掌握着总线的控制权，可对涉及的外设端口或存储器单元进行读/写操作。

② 从态方式。在从态方式时，CPU 是总线的控制者，而 DMA 控制器不过是一个普通的外部设备，有若干个端口而已，它的地位同一般的 I/O 接口芯片是一样的。

### 2. 传送方式

8237A 通过编程，可选择以下四种传送方式。

① 单字节传送方式。单字节传送方式时，一次只传送一个字节，然后释放总线。若又有外设 DMA 请求，8237A 再向 CPU 发下一次总线请求 HRQ；获得总线控制权后，再传送下一个字节数据。

② 数据块传送方式。数据块传送方式时，响应一次 DMA 请求，将完成设定的字节数的全部传送。当字节数计数器减 1 由 0 变为 FFFFH 时，产生 TC 有效信号，使 8237A 将总线控制权交还给 CPU，从而结束 DMA 操作方式。外部有效的  $\overline{\text{EOP}}$  信号也可以终结 DMA 传送。

③ 请求传送方式。请求传送方式又称查询方式，类似数据块传送，但每传送一个字节后，会检测 DREQ 状态：若无效则停止，若有效则继续 DMA 传送。在下述情况之一发生时，将停止传送：DREQ 变为无效；字节数计数器减 1 由 0 变为 FFFFH，产生 TC 信号；外界输入  $\overline{\text{EOP}}$  有效信号。

④ 级联方式。这种方式允许连接一个以上的芯片来扩展 DMA 通道的个数。

### 3. 传送类型

DMA 系统无论是工作在单片方式，还是多片级联；也不管是采用单字节传送、数据块传送，还是请求哪种传送方式，都可以对每个通道的方式寄存器进行设置。采用以下三种不同的传送类型。

① DMA 读。8237A 输出有效的  $\overline{\text{MEMR}}$  和  $\overline{\text{IOW}}$  信号，把存储器的数据读到 I/O 设备。

② DMA 写。8237A 输出有效的  $\overline{\text{IOR}}$  和  $\overline{\text{MEMW}}$  信号，把 I/O 设备的数据写到存储器。

③ DMA 校验。这是一种伪传输，实际上是校验 8237A 芯片内部的读/写功能。在这种传送类型中，8237A 芯片的操作如同 DMA 读和 DMA 写一样，产生地址信号及对  $\overline{\text{EOP}}$  响应等，但存储器和 I/O 设备的控制线（ $\overline{\text{MEMR}}$ ， $\overline{\text{MEMW}}$ ， $\overline{\text{IOW}}$ ， $\overline{\text{IOR}}$ ）均处于无效状态，禁止实际传送。

### 4. 优先级编码

8237A 芯片可设定为两种优先级编码：固定优先级和循环优先级。

固定优先级中四个通道的优先级顺序是固定的，DREQ<sub>0</sub> 最高，DREQ<sub>3</sub> 最低。循环优先级中四个通道的优先级顺序是可变的，但其变化仍有一定的规律。当某一个通道申请 DMA



请求并被响应服务后,它就被指定为最低优先级,它的下一级就成为最高优先级。值得注意的是,无论在什么情况下,DMA 请求禁止嵌套服务。当一个通道的 DMA 请求被响应并服务后,其他三个通道的 DMA 请求都将被禁止,无论它们的优先级是高还是低。优先权排队只在 DMA 响应之前有效,DMA 响应之后则无效。

### 5. 工作周期

8237A 有两类工作周期:空闲周期和操作周期。操作周期也叫有效周期。全部工作周期分为七种时钟状态(时钟周期):空闲状态  $S_1$ ,起始状态  $S_0$ ,传送状态  $S_1$ 、 $S_2$ 、 $S_3$ 、 $S_4$  及等待状态  $S_w$ 。

(1) 空闲周期。在没有 DMA 请求时,8237A 就处于空闲周期,执行连续的空闲状态  $S_1$ 。在空闲周期内,每个  $S_1$  状态都要进行两种检测:① 检测有无  $\overline{CS}$  信号,以确定有无 CPU 对 8237A 的操作要求。若  $\overline{CS}$  信号有效,则表明 CPU 要对 8237A 进行读/写操作,此时,8237A 以从态方式工作,作为 CPU 的一个 I/O 接口,进入编程状态,受 CPU 的控制。对 8237A 的 DMA 初始化工作就是在这种状态下实现的。② 检测有无 DREQ 信号,以确定是否有 I/O 设备送来有效的 DMA 请求。若 DREQ 有效,则产生总线请求 HRQ,从  $S_1$  状态进入  $S_0$  状态,此时,8237A 以主态方式工作,以后的工作都由 8237A 来控制。

(2) 操作周期。8237A 在空闲周期内检测出 DMA 请求 DREQ 有效后,便进入操作周期(有效周期)。操作周期有 6 种状态: $S_0$  起始状态、 $S_1$  工作状态 1、 $S_2$  工作状态 2、 $S_3$  工作状态 3、 $S_4$  工作状态 4、 $S_w$  等待状态。

### 6. 读/写时序

DMA 控制器可在两种操作时序中任选一种:正常时序和压缩时序。读/写时序方式的实质是,控制读/写脉冲发出的时间与时钟信号 CLOCK 的对应关系。

① 正常时序。正常时序传送一个字节的的数据包含 4 个时钟脉冲周期,即  $S_1 \sim S_4$  状态。产生的读/写脉冲信号与这 4 个状态有确定的对应关系。若数据块传送中不改变高 8 位地址,则省去  $S_1$ ,只占用  $S_2$ 、 $S_3$ 、 $S_4$  3 个时钟周期。

② 压缩时序。压缩时序方式时所占用的脉冲数将减少。压缩时序操作把读命令的宽度压缩到等于写命令的宽度,省掉了  $S_3$ ,即由  $S_4$  完成读和写的操作。所以,在一段压缩时序方式下传送一个字节的的数据需要占用 3 个时钟周期,即  $S_1$ 、 $S_2$ 、 $S_4$ 。在大多数情况下高 8 位地址并不改变,于是省掉了  $S_1$ ,因此,在数据块传送中大多数情况下占用 2 个时钟周期,即  $S_2$  和  $S_4$ 。在系统性能允许的范围内,压缩时序能获得较高的数据吞吐量。

### 7. 自动初始化方式

通过对方式字寄存器的编程,可设置某个通道为自动初始化方式。

自动初始化方式的功能是,当该通道完成一个数据传送并产生  $\overline{EOP}$  信号时(可能是由内部的 TC 产生,也可能由外部产生),用基地址寄存器和基字节数计数器的内容,使相应的当前地址寄存器和当前字节数计数器恢复初值。当前地址寄存器和当前字节数计数器的最初值,是由 CPU 在初始化编程时写入的(这个最初值同时也写入到基地址寄存器和基字节数计数器),但在 DMA 传送过程中,当前地址寄存器和当前字节数计数器的内容被不断地修改,而基地址寄存器和基字节数计数器的内容维持不变(除非重新编程)。在自动初始化以后,通道就做好了进行另一次 DMA 传送的准备。

## 8. 存储器到存储器的传送

利用存储器到存储器传送方式，可以使数据块从一个存储空间传送到另一个存储空间，将程序的影响和传输时间减到最小。

这种方式需要占用 8237A 的 2 个通道。由通道 0 的地址寄存器提供源地址，通道 1 的地址寄存器提供目的地址。通道 1 的字节数计数器编程为要传送的字节数，传送由设置通道 0 的软件 DREQ 启动。8237A 按正常方式向 CPU 发出 HRQ，当 CPU 发出总线响应信号 HLDA 后，DMA 传送即可开始。

### 6.4.4 8237A的编程

依靠 8237A 的可编程特性实现各种工作方式的选择和设定。8237A 在 HLDA 信号处于无效的任何时间里，即使 HRQ 有效，也可以接受 CPU 对它的编程。CPU 对 8237A 的编程初始化工作是通过 8237A 的端口进行的。8237A 的端口是用低 4 位地址线  $A_3A_2A_1A_0$  编址的，共有 16 个端口地址，其具体编址情况如表 6-1 所示。假设我们以 DMA 代表 16 个端口地址的首地址，那么写通道 2 基字节数计数器的端口地址可表示为  $DMA+05H$ ，写方式寄存器的端口地址可表示为  $DMA+0BH$ 。16 个端口地址分为以下两部分。

(1) 00H~07H 分配给 4 个通道的相应的 16 位寄存器，它们的端口地址分配如下：

	基和当前地址寄存器端口	基和当前字节数计数器端口
通道 0	DMA+0	DMA+1
通道 1	DMA+2	DMA+3
通道 2	DMA+4	DMA+5
通道 3	DMA+6	DMA+7

由于这几个寄存器都是 16 位的，而 8237A 只有 8 位数据线，因此，读/写操作均分两次进行（使用两条 I/O 指令）。每次读/写前应先应用软件命令清除字节指针触发器（即先/后触发器），然后，以先低字节后高字节的顺序进行读/写操作。

(2) 08H~0FH 分配给其他寄存器，其中包括 3 条不使用数据总线而只利用端口地址进行操作的清除命令。

8237A 的初始化编程命令字总共有 8 个，其中控制命令有 5 个：方式字、命令字、请求字、屏蔽字、状态字；清除命令有 3 个：清除字节指针触发器、主清除命令、清除屏蔽寄存器。

#### 1. 控制命令

① 方式字写入端口地址 0BH，其主要功能是：选择传送方式和传送类型，设置自动初始化方式和地址增量方向。

② 命令字写入端口地址 08H，其主要功能是：选择 DREQ、DACK 有效极性，读/写时序，优先级编码方式等。

③ 请求字写入端口地址 09H，其主要功能是：发生软件 DMA 请求。

④ 屏蔽字写入端口地址 0AH 或 0FH，其主要功能是：允许或禁止通道的 DMA 请求。

⑤ 状态字从端口地址 08H 读出，其主要功能是：反映通道 DMA 请求状态和是否有 TC 信号。

8237A 控制命令字格式如图 6-15 所示。

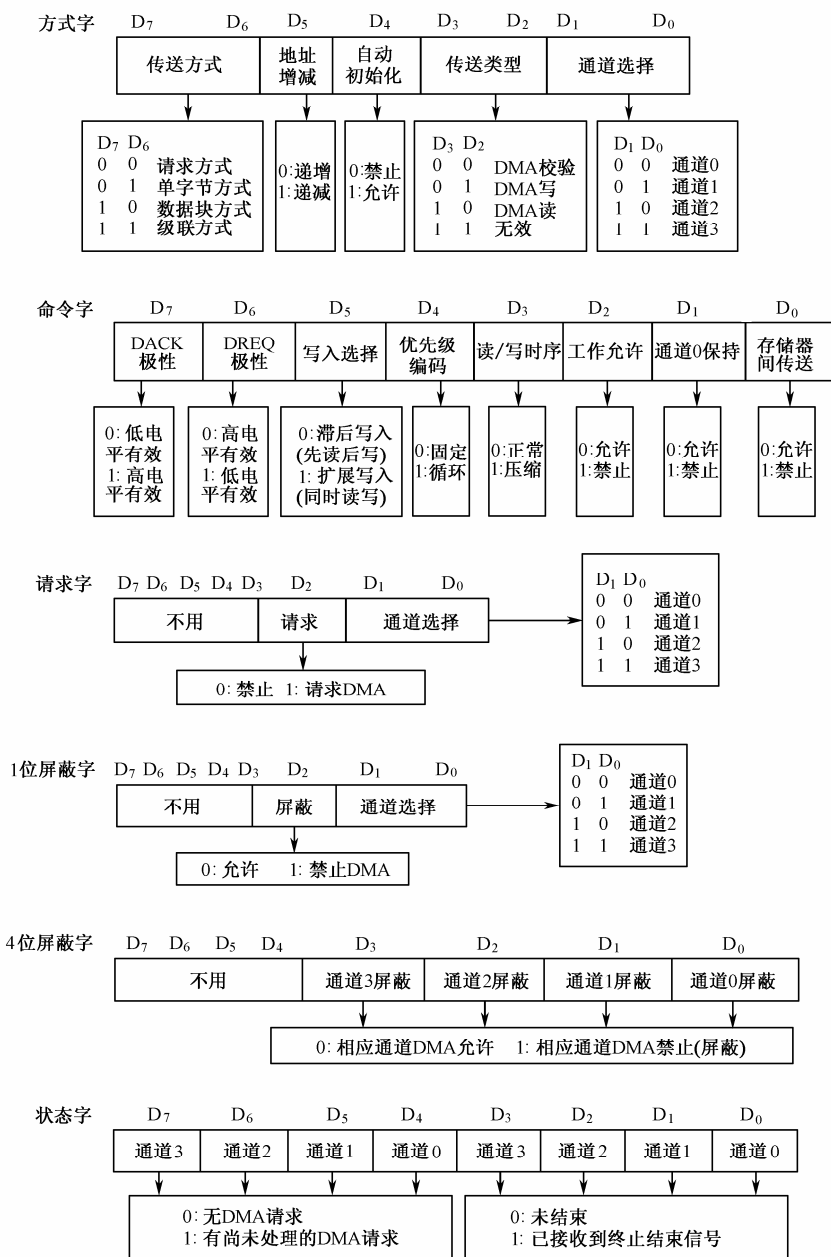


图 6-15 8237A 控制命令字格式

## 2. 清除命令

清除命令有 3 个。这 3 个命令与数值无关，不需通过数据总线。即执行输出指令时，AL 的内容可任意设置，只要对特定的端口地址执行一次写操作，依靠这个地址和控制信号，命令就生效。这就是所谓的“软命令”。

① 清除字节指针命令（写入端口地址 0CH）。即清除高/低触发器命令，或叫清除先/后触发器命令，是专为 16 位寄存器的读/写而设置的。因为数据线是 8 位的，所以 16 位数据

要分两次读/写,而且要使用同一个端口地址。为区分两个高低字节,8237A 设置了先/后触发器作为字节指针,为“0”时对应低字节,为“1”时对应高字节。而且每次对 16 位寄存器的读/写操作都将使字节指针自动翻转一次:由“0”变“1”,或由“1”变“0”,从而控制高、低字节的读/写,系统复位后,字节指针被清“0”。

清除字节指针命令使字节指针(先/后触发器)被清“0”。

② 主清除命令(写入端口地址 0DH)。此命令与硬件的 RESET 信号功能相同,是软件复位命令,除使屏蔽寄存器各位置“1”外,其他各寄存器均被清“0”,使 8237A 进入空闲状态,准备接收 CPU 对它的初始化编程,此时 8237A 处于从态方式。

③ 清除屏蔽寄存器命令(写入端口地址 0EH)。其功能是将 4 个通道的屏蔽位清除,允许它们接受 DMA 请求。该命令是对屏蔽寄存器操作的第三个命令。

### 3. 8237A的编程步骤

8237A 的编程步骤如下:

- ① CPU 发主清除命令(复位);
- ② 写入基地址及当前地址的值;
- ③ 写入基字节数和当前字节数的初值;
- ④ 写入方式字;
- ⑤ 写入屏蔽字;
- ⑥ 写入命令字;

⑦ 写入请求字,可用软件 DMA 请求启动通道,也可在①~⑥完成以后,等待外部 DREQ 请求信号。

在编程中应注意以下几点:

① 16 位寄存器的读/写顺序:先低后高。最好先清除字节指针,使先/后触发器清“0”,保证先低后高的顺序。

② 保证编程和 HLDA 是互相排斥的。编程时应保证 CPU 的编程(对 8237A 的初始化写入操作)与 HLDA 是互相排斥的。因为当 CPU 正对某个通道编程的同时,若该通道接收到一个 DMA 请求,问题就会发生。例如,CPU 正开始对通道 0 的 16 位寄存器操作,此时该通道收到一个 DMA 请求,若允许 8237A 工作,且通道 0 是非屏蔽的,则在写入一个字节后就开始了 DMA 服务。为确保硬件信号不影响软件编程,应在编程开始时禁止 8237A 工作,或屏蔽掉该通道,等到编程结束后再允许 8237 工作或清除通道的屏蔽位。

【例 6-1】8237A 数据块传送。设在某 8088 系统中,用 8237A 通道 1 将内存 1000H 单元开始的 24KB 数据转存到软盘中(暂不考虑 20 位地址的问题,可认为 1000H 就是基地址的初值),采用数据块方式传送,地址增量方式,只传送一遍,设 DREQ 和 DACK 低电平有效,当  $A_{15} \sim A_4$  为 000000000111 时选中 8237A。要求设计 8237A 通道 1 的初始化程序。

① 端口地址。 $A_3 \sim A_0$  由 8237A 芯片内部译码,编码范围为 0000~1111,再与  $A_{15} \sim A_4$  组合,则端口地址范围是 0070H~007FH。

② 传送字节数。24KB 对应的 16 进制数为 6000H,但写入通道字节数计数器的值应为:6000H - 1 = 5FFFH。因为 TC 的产生不是在计数器由 1 到 0 的跳变处,而是在计数器由 0 到 FFFFH 的跳变处,所以写入的计数初值应比实际字节数少一个。

- ③ 方式字。按题目要求，控制字的组合为：1000 1001B。
- ④ 一位屏蔽字。按题目要求，一位屏蔽字的组合为：0000 0001B。
- ⑤ 命令字。按题目要求，命令字的组合为：0100 0000B。
- ⑥ 初始化程序如下。

```
START: MOV  DX, 007DH      ; 发主清除命令
        OUT  DX, AL
        MOV  DX, 0072H
        MOV  AL, 00H
        OUT  DX, AL        ; 送基地址和当前地址低 8 位
        MOV  AL, 10H
        OUT  DX, AL        ; 送基地址和当前地址高 8 位
        MOV  DX, 0073H
        MOV  AL, 0FFH      ; 送基值和当前计数值低 8 位
        OUT  DX, AL
        MOV  AL, 5FH       ; 送基值和当前计数值高 8 位
        OUT  DX, AL
        MOV  DX, 007BH
        MOV  AL, 89H       ; 写入方式控制字
        OUT  DX, AL        ; DMA 读传送
        MOV  DX, 007AH
        MOV  AL, 01H       ; 写入屏蔽字
        OUT  DX, AL
        MOV  DX, 0078H
        MOV  AL, 40H       ; 写入命令控制字
        OUT  DX, AL
```

## 习题

1. 外设的含义是什么？
2. CPU 与外设交换信息，通常需要哪些信号？
3. I/O 端口的编址方式有几种？各有什么特点？8088 系统的 I/O 端口编址方式属于哪种？
4. CPU 与外设之间的数据传送方式有几种？它们各自的功能及接口电路的特点是什么？
5. 设计一个查询式输入接口电路，并根据这一电路编写相应的查询输入程序。
6. DMA 控制器占用总线有哪几种方法？
7. 8237A 的基本功能是什么？
8. 8237A 有几种传送方式？有几种传送类型？
9. 什么是 8237A 的“软命令”？软命令有什么特点？举例说明。

# 第7章

## 中 断 系 统

### 教学目的和要求

本章主要介绍中断的基本概念，中断响应过程，8088 的中断方式，IBM PC/XT 的中断方式，中断控制器 8259。重点掌握中断的概念、中断响应过程及 8088 的中断方式。

所谓中断，是指 CPU 在正常运行程序时，由程序预先安排好的事件，或者由内、外部事件引起 CPU 中断正在运行的程序，而转到为预先安排的事件或内、外部事件服务的程序中去。这些引起程序中断的事件称为中断源。预先安排好的事件是指 PC 的中断指令。执行到此，立即转到对应的服务程序去执行。内部事件是指系统板上出现的一些事件信号，中断指令也可看做内部事件。外部事件是指某些接口设备所发出的请求中断程序执行的信号，这些信号称为中断请求信号。中断请求信号何时发生是不可预知的，然而，它们一旦请求中断，就会向 CPU 发出电信号，因此这些信号 CPU 是可以测知的。这样，CPU 就无需花大量的时间去查询这些信号。例如，键盘何时键按下是随机的，因而 CPU 可以对键盘不加等待，而去执行其他程序，一旦有键按下，键盘马上产生中断信号，CPU 得知这一信号后，就立即去执行为键盘服务的中断程序，服务结束后，CPU 又恢复执行被中断了的程序。中断服务程序执行完，返回原来执行程序的中断处（称为断点）继续往下执行，称为中断返回。有时中断请求信号（即中断源）可能有好几个，因此 CPU 响应这些中断就得有先后次序，这称为中断的优先级。优先级高的中断，CPU 首先响应；优先级低的中断暂不响应，称为挂起。有些中断源产生的中断，可以用编程的方法，使 CPU 不予理睬，这叫做中断的屏蔽。CPU 响应中断转去执行中断服务程序前，需要把被中断程序的现场信息保存起来，以便执行完中断服务程序后，接着从被中断程序的断点处继续往下执行。现场信息包括程序计数器的内容、CPU 的状态信息、执行指令后的结果特征和一些通用寄存器的内容。有些信息的保存，如程序计数器的内容等，由机器硬件预先安排完成，称为中断处理的隐操作；而有些信息的保存是在中断服务程序中预先安排的。CPU 响应中断，由中断源提供地址信息，引导程序转移并执行中断服务程序，这个地址信息称为中断向量，它一般是和中断源相对应的。PC 采用类型码来标识中断源。

在数据采集或实时控制中，CPU 对接口设备的控制或交换信息，可采用查询、中断、DMA 等方式，而中断方式以其执行速度快，可实时处理，不占用 CPU 过多的时间等优点，在接口技术中较多地被采用。

## 7.1 中断概述

### 7.1.1 中断的必要性

如上所述，CPU 在与外设交换信息时，若用查询的方式，则 CPU 就要浪费很多时间去等待外设。这样就存在一个快速的 CPU 与慢速的外设之间的矛盾，这也是计算机在发展过程中遇到的严重问题之一。为解决这个问题，一方面要提高外设的工作速度；另一方面发展了中断的概念。中断有以下好处。

#### 1. 同步操作

有了中断功能，就可以使 CPU 和外设同时工作。CPU 在启动外设工作后，就继续执行主程序，同时外设也在工作；当外设把数据准备好后，发出中断申请，请求 CPU 中断它的程序，执行输入或输出（中断处理）；处理完以后，CPU 恢复执行主程序，外设也继续工作。而且有了中断功能，CPU 可命令多个外设同时工作，这样就大大提高了 CPU 的利用

率，也提高了输入、输出的速度。

## 2. 实现实时处理

当计算机用于实时控制时，中断是一个十分重要的功能。现场的各种参数、信息，需要的话可在任何时间发出中断申请，要求 CPU 处理；CPU 可以马上响应（若中断是开放的话）并加以处理。这样的及时处理在查询工作方式下是做不到的。

## 3. 故障处理

计算机在运行过程中，往往会出现事先预料不到的情况，或出现一些故障，如电源突跳、存储器出错、运算溢出等，计算机可以利用中断系统自行处理，不必停机或报告工作人员。

### 7.1.2 中断源

引起中断的原因，或能发出中断申请的来源，称为中断源。通常中断源有以下几种。

#### 1. 一般的输入/输出设备

如键盘、纸带读入机、打印机等。

#### 2. 数据通道中断源

如磁盘、磁带等。

#### 3. 实时时钟

在控制中，常遇到时间控制问题等，若用 CPU 执行一段程序来实现延时的方法，则在这段时间内，CPU 不能干别的工作，降低了 CPU 的利用率。所以，常采用外部时钟电路，当需要定时时，CPU 发出命令，令时钟电路（电路的定时时间通常是可编程的，即可用程序来确定和改变）开始工作，待规定的时间到了后，时钟电路发出中断申请，由 CPU 加以处理。

#### 4. 故障源

例如，电源掉电时，需要把正在执行的程序的状态——PC（或 IP）、各个寄存器的内容和标志位的状态保留下来，以便重新供电后能从断点处继续运行。另外，目前绝大部分微型计算机中，RAM 为半导体存储器，电源掉电后，必须接入备用电源（电池）供电，以保护存储器中的信息。所以，在直流电源上并上大电容，使其因掉电或电压下降到一定值时就发出中断申请，由计算机的中断系统执行上述的各项操作。

#### 5. 为调试程序而设置的中断源

一个新程序编制好以后，必须经过反复调试才能可靠地工作。在程序调试时，为了检查结果，或为了寻找毛病所在，往往要求在程序中设置断点，或进行单步工作（一次只执行一条指令），这要由中断系统来实现。

### 7.1.3 中断系统的功能

为了满足上述各种情况下的中断要求，中断系统应具有如下功能。

#### 1. 实现中断及返回

当某一中断源发出中断申请时，CPU 能决定是否响应这个中断请求（当 CPU 在执行更紧急、更重要的工作时，可以暂不响应中断）。若允许响应这个中断请求，CPU 必须在现行



的指令执行完后，把断点处的 IP 和 CS 的值（即下一条应执行指令的地址）、各个寄存器的内容和标志位的状态，推入堆栈保留下来——保护断点和现场。然后才能转到需要处理的中断源的服务程序（Interrupt Service Routine）的入口，同时清除中断请求触发器。当中断处理完后，恢复被保留下来的各个寄存器和标志位的状态（称为恢复现场），并恢复 IP 和 CS 的值（称为恢复断点），使 CPU 返回断点，继续执行主程序。

## 2. 能实现优先排队

通常，在系统中有多中断源，会出现两个或更多个中断源同时提出中断请求的情况，这样就要求设计者事先根据轻重缓急，给每个中断源确定一个中断级别——优先权。当多个中断源同时发出中断申请时，CPU 能找到优先权级别最高的中断源，响应它的中断请求；在优先权级别最高的中断源处理完以后，再响应级别较低的中断源。

## 3. 高级中断源能中断低级的中断处理

当 CPU 响应某一中断源的请求，在进行中断处理时，若有优先权级别更高的中断源发出中断申请，则 CPU 要能中断正在进行的中断服务程序，保留这个程序的断点和现场（类似于子程序嵌套），响应高级中断，在高级中断处理完以后，再继续执行被中断的中断服务程序。当发出新的中断申请的中断源的优先级别与正在处理的中断源为同级或更低时，CPU 就先不响应这个中断申请，直至正在处理的中断服务程序执行完以后才去处理新的中断申请。

# 7.2 CPU响应中断的条件和过程

由于引脚的限制，CPU 的中断请求线的数量是有限的。例如，8088 只有一条可屏蔽硬件中断请求线（INTR）。最简单的情况当然是只有一个中断源，我们就从这个最简单的情况分析。

## 7.2.1 CPU响应中断的条件

### 1. 设置中断请求触发器

每一个中断源，要能发出中断请求信号，且这个信号能一直保持到 CPU 响应这个中断请求后，才可清除中断请求。故要求每一个中断源有一个中断源请求触发器 A，如图 7-1 所示。

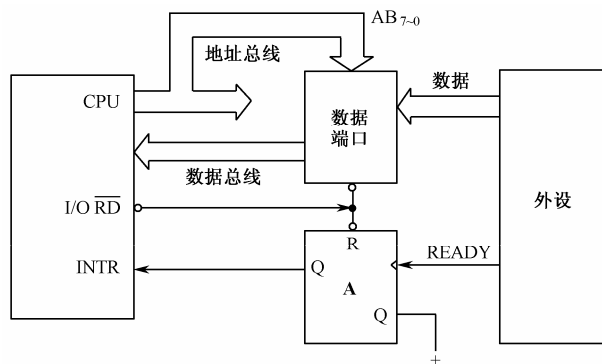


图 7-1 设置中断请求的电路

## 2. 设置中断屏蔽触发器

在实际系统中，往往有多个中断源。为了增强控制的灵活性，需要在每一个外设的接口电路中增加一个中断屏蔽触发器 B，只有当此触发器的输出为“1”时，外设的中断请求才能被送至 CPU，如图 7-2 所示。可把 8 个外设的中断屏蔽触发器组成一个端口，用输出指令来控制它们的状态。

## 3. 中断是开放的

在 CPU 内部有一个中断允许触发器。只有当其为“1”时（即中断开放时），CPU 才能响应中断；若其为“0”（即中断是关闭的），则即使 INTR 线上有中断请求，CPU 也不响应。这个触发器的状态可由 STI 和 CLI 指令改变。当 CPU 复位时，中断允许触发器的输出为“0”，即关中断，所以必须用 STI 指令开中断。当中断响应后，CPU 自动关中断，所以必须在中断服务程序中用 STI 指令开中断。

## 4. CPU在现行指令结束后响应中断

CPU 在现行指令运行到最后一个机器周期的最后一个 T 状态时，才采样 INTR 线。若发现有中断请求，则把内部的中断锁存器置“1”，下一个机器周期不进入取指周期，而进入中断周期。其时序流程如图 7-3 所示。

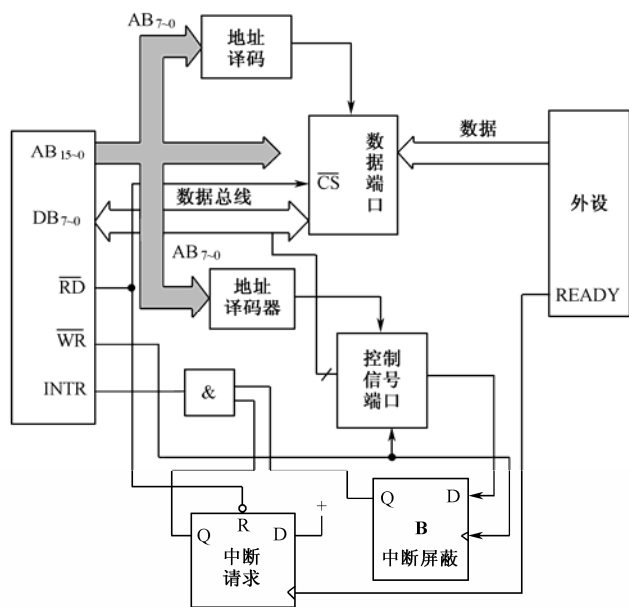


图 7-2 具有中断屏蔽的接口电路

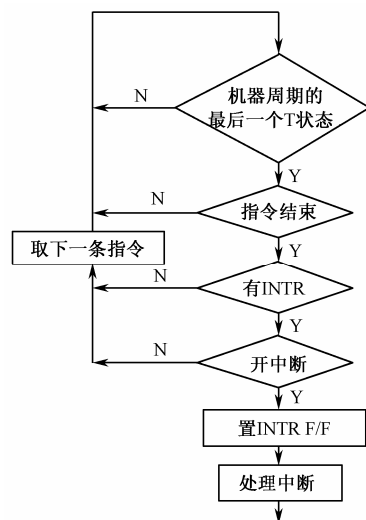


图 7-3 中断时序流程图

## 7.2.2 CPU对中断的响应

当满足上述条件后，CPU 响应中断，转入中断周期，CPU 做以下几件事。

### 1. 关中断

8088 在 CPU 响应中断后，发出中断响应信号 INTA 的同时，内部自动地实现关中断。

## 2. 保留断点

CPU 响应中断，封锁 IP+1，且把 IP 和 CS 推入堆栈保留，以备中断处理完毕后，能返回主程序。

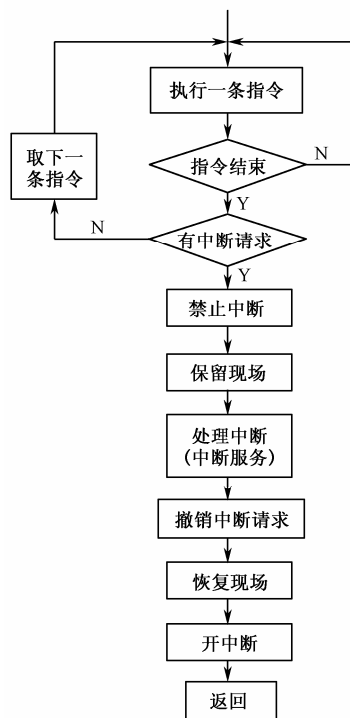


图 7-4 中断响应、服务及返回流程图

## 3. 保护现场

为了使中断处理程序不影响主程序的运行，故要把断点处有关的各个寄存器的内容和标志位的状态，推入堆栈保护起来。8088 通过软件（即在中断服务程序中）把要用到的寄存器的内容用 PUSH 指令推入堆栈。

## 4. 给出中断入口，转入相应的中断服务程序

8088 由中断源提供的中断矢量形成中断入口地址（即中断服务程序的起始地址）。在中断服务程序完成后，还要做下述的 5、6 两步工作。

## 5. 恢复现场

把所保存的各个内部寄存器的内容和标志位的状态，从堆栈弹出，送回 CPU 中的原来位置。这个操作在 8088 中也是通过服务程序中的 POP 指令来完成的。

## 6. 开中断与返回

在中断服务程序的最后，要开中断（以便 CPU 能响应新的中断请求）和安排一条返回指令，将堆栈内保存的 IP 和 CS 的值弹出，程序恢复到主程序中运行。

上述过程可用如图 7-4 所示的流程图表示。

# 7.3 中断优先权及多重中断

## 7.3.1 中断优先权

如前所述，实际的系统中，是有多个中断源的，但是，由于 CPU 引脚的限制，往往只有一条中断请求线。于是，当有多个外中断源同时请求时，CPU 就要识别出哪些中断源有中断请求，辨别和比较它们的优先权（Priority），先响应优先权级别最高的中断请求。另外，当 CPU 处理中断时，也要能响应更高级的中断请求，而屏蔽掉同级或较低级的中断请求。

要判别和确定各个中断源的优先权可以用软件和硬件两种方法。

### 1. 用软件确定中断优先权

软件采用查询技术。当 CPU 响应中断后，就用软件查询以确定是哪些外设申请了中断，并判断它们的优先权。

把 8 个外设的中断请求触发器组合起来，作为一个端口，并赋以设备号，如图 7-5 所

示, 把各个外设的中断请求信号相“或”后, 作为 INTR 信号, 故任一外设中有中断请求, 都可向 CPU 送出 INTR 信号。当 CPU 响应中断后, 把中断寄存器的状态, 作为一个外设读入 CPU, 逐位检测它们的状态, 若有中断请求就转到相应的服务程序的入口。其流程如图 7-6 所示。

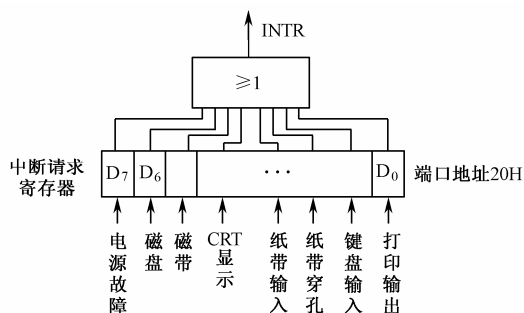


图 7-5 用软件查询方式实现的接口电路

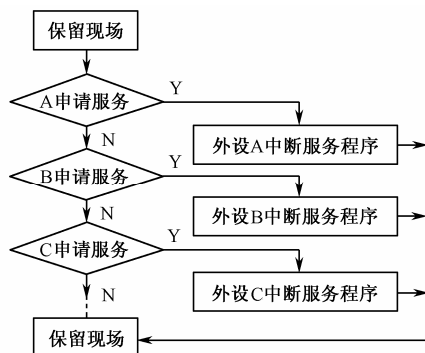


图 7-6 软件查询程序流程图

查询程序有两种实现方式

#### (1) 屏蔽法

```
IN      AL, [20H]    ; 输入中断请求触发器的状态
TEST    AL, 80H      ; 检查最高位
JNE     PWF          ; 外设 A 中断服务程序
TEST    AL, 40H      ; 检查次高位
JNE     DISS         ; 外设 B 中断服务程序
TEST    AL, 20H      ; 检查最低位
JNE     MT           ; 外设 C 中断服务程序
.....
```

#### (2) 移位法

```
XOR     AL, AL
IN      AL, [20H]    ; 输入中断请求触发器的状态
RCL     AL, 1
JC      PWF          ; 外设 A 中断服务程序
RCL     AL, 1
JC      DISS         ; 外设 B 中断服务程序
.....
```

查询技术的优点是:

① 询问的次序, 即是优先权的次序。显然, 最先询问的, 优先权的级别最高。

② 省硬件。不需要有判断与确定优先权的硬件排队电路。

缺点是: 由询问转至相应的服务程序入口的时间长, 尤其是在中断源较多的情况下。

## 2. 硬件优先权排队电路

### (1) 中断优先权编码电路

采用硬件编码器和比较器的优先权排队电路，如图 7-7 所示。

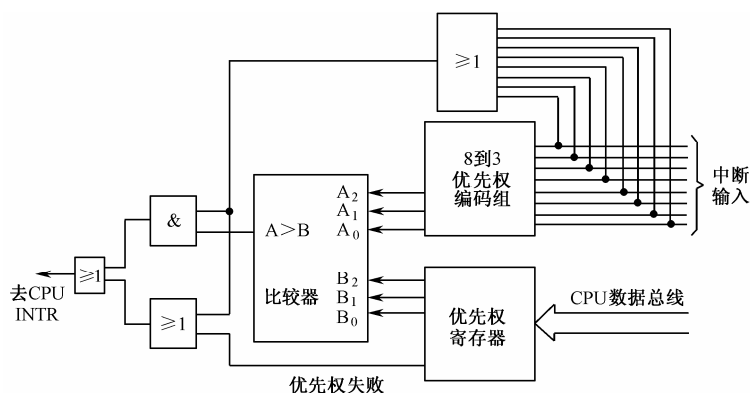


图 7-7 编码器和比较器的优先权排队电路

若有 8 个中断源，当任意一个有中断请求时，通过或门，即可有一个中断请求信号产生，能否送至 CPU 的中断请求线，还要受比较器的控制（若优先权失效信号为低电平，则与门 2 关闭）。

8 条中断输入线的任一条，经过编码器可以产生三位二进制优先权编码  $A_2A_1A_0$ ，优先权最高的线的编码为 111，优先权最低的线的编码为 000。而且若有多个输入线同时输入，则编码器只输出优先权最高的编码。

正在进行中断处理的外设的优先权编码，通过 CPU 的数据总线，送至优先权寄存器，然后输出编码  $B_2B_1B_0$  至比较器，以上过程是由软件实现的。

比较器比较编码  $A_2A_1A_0$  与  $B_2B_1B_0$  的大小，若  $A < B$ ，则“ $A > B$ ”端输出低电平，封锁与门 1，就不向 CPU 发出新的中断申请（即当 CPU 正在处理中断时，当有同级或低级的中断源申请中断时，优先权排队电路就屏蔽掉它们的请求）；只有当  $A > B$  时，比较器输出端才为高电平，打开与门 1，将中断请求信号送至 CPU 的 INTR 输入端，CPU 就中断正在进行的中断处理程序，转去响应更高级的中断。

若 CPU 不再进行中断处理时（即在执行主程序），则优先权失效信号为高电平，当有一种中断源请求中断时，都能通过与门 2 发出 INTR 信号。这样的优先权电路，如何才能转入优先权最高的外设的服务程序的入口呢？当外设的个数小于等于 8 时，则它们公用一个产生中断矢量的电路，根据比较器的编码  $A_2A_1A_0$ ，就能做到不同的编码转入不同的入口地址。

### (2) 雏菊花环（Daisy Chain）式或称为链式优先权排队电路

当多个输入有中断请求时，则由中断输入信号的“或”电路产生 INTR 信号，送至 CPU。CPU 执行完现行指令后，响应中断，发出中断响应信号。但究竟响应哪一个中断呢？或 CPU 是转向哪一个中断服务程序的入口呢？这要由图 7-8 所示的链式优先权排队电路确定。

当中断响应为高电平时,若 F/F A 有中断请求,则它的输出为高电平,于是与门 A<sub>1</sub> 输出为高电平,由它控制转至中断 1 的服务程序的入口,且门 A<sub>2</sub> 输出为低电平,因而使门 B<sub>1</sub>、B<sub>2</sub>、C<sub>1</sub>、C<sub>2</sub>…所有下面各级门的输入和输出全为低电平,即屏蔽了以下的各级。

若第一级没有中断请求,即 F/F A=0,则中断输出为低电平,但门 A<sub>2</sub> 的输出却为高电平,即把中断响应传递至中断请求 2。若此时 F/F B=1,则与门 B<sub>1</sub> 输出为高电平,控制转去执行中断 2 的服务程序;此时与门 B<sub>2</sub> 的输出为低电平,因而屏蔽了以下各级。而若 F/F B=0,则与门 B<sub>1</sub> 输出为低电平,而与门 B<sub>2</sub> 输出为高电平,把中断响应传递至中断请求 3……

综上所述,在链式优先权排队电路中,若上级的输出信号为“0”,则屏蔽了本级和所有的低级中断;若上级输入为“1”,在本级有中断请求时,则转去执行本级的处理程序,且使本级至下级的输出为“0”,屏蔽所有低级中断;若本级没有中断请求,则本级至下级的输出为“1”,允许下一级中断,故在链的最前面的优先权最高。

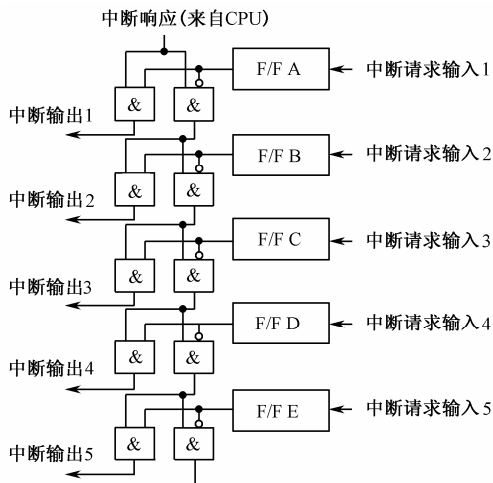


图 7-8 链式优先权排队电路

### 7.3.2 多级中断的概念

多级中断系统是指计算机系统中有相当多的中断源,根据各中断事件的轻重缓急而分成若干级别,每一中断级分配一个优先权。一般来说,优先权高的中断级可以打断优先权低的中断服务程序,以程序嵌套方式进行工作。如图 7-9 (a) 所示,三级中断优先权高于二级,而二级中断优先权高于一级。

根据系统的配置不同,多级中断又可分为一维多级中断和二维多级中断,如图 7-9 (b) 所示。一维多级中断是指每一中断中只有一个中断源,而二维多级中断是指每一级中断中又有多多个中断源。图中虚线左边结构为一维多级中断,如果取掉虚线则成为二维多级中断结构。

对于多级中断,我们着重说明如下几点:

第一,一个系统若有  $n$  级中断,在 CPU 中就有  $n$  个中断请求触发器,总称为中断请求寄存器;与之对应的有  $n$  级中断屏蔽触发器,总称为中断屏蔽寄存器。与单级中断不同,在多级中断中,中断屏蔽寄存器的内容是一个很重要的程序现场,因此在响应中断时,需要把中断屏蔽寄存器的内容保存起来,并设置新的中断屏蔽状态。一般在某一级中断被响应后,要置“0”(关闭)本级和优先权低于本级的中断屏蔽触发器,置“1”(开放)更高级的中断屏蔽触发器,以此来实现正常的中断嵌套。

第二,多级中断中的每一级可以只有一个中断源,也可以有多个中断源。在多级中断之间可以实现中断嵌套,但是同一级内的不同中断源的中断是不能嵌套的,必须是处理完一个中断后再响应和处理同一级内的其他中断源。

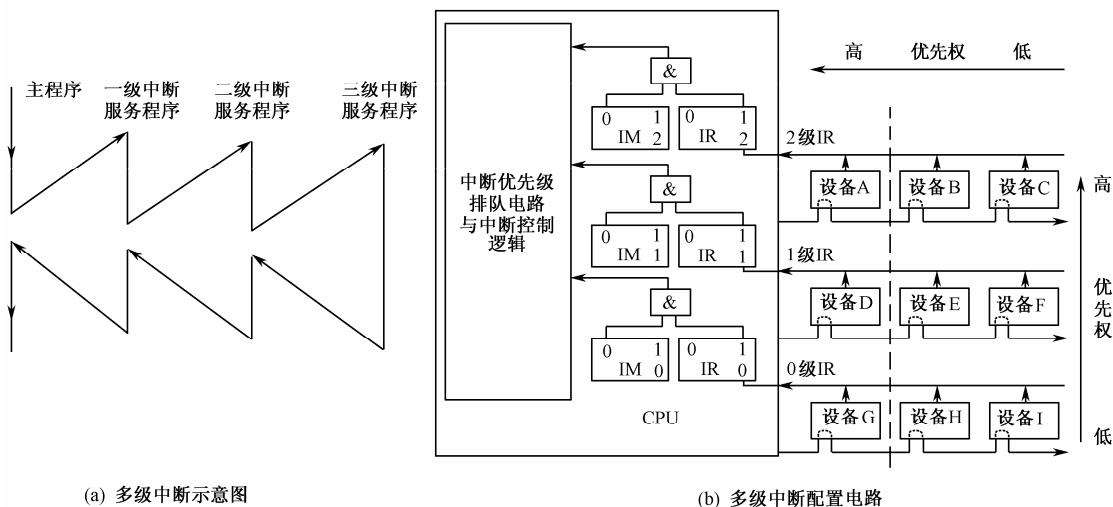


图 7-9 多级中断

第三，设置多级中断的系统一般都希望有较快的中断响应时间，因此首先响应哪一级中断和哪个中断源，由硬件逻辑实现，而不是由程序实现。图 7-9 (b) 中的中断优先级排队电路，就是用于决定优先响应中断级的硬件逻辑。另外，在二维中断结构中，除了由中断优先级排队电路确定优先响应中断级外，还要确定优先响应的中断源，一般通过链式查询的硬件逻辑来实现。显然，这是用独立请求方式与链式查询方式相结合的方法来决定首先响应哪个中断源。

第四，和单级中断情况类似，在多级中断中也使用中断堆栈保存现场信息。使用堆栈保存现场的好处是：① 控制逻辑简单，保存和恢复现场的过程按先进后出顺序进行。② 每一级中断不必单独设置现场保护区，各级中断现场可按其顺序放在同一栈里。

## 7.4 8088 的中断方式

8088 有两种类型的中断：由执行某些指令引起的软中断和设备引起的硬中断，这两类中断均有中断类型码相对应。

### 1. 软中断

执行下述指令时，将产生或可能产生中断。

#### (1) DIV (除) 或 IDIV (整除) 指令

当执行这些除法指令时，若除数为 0 或商溢出，则产生中断，称为 0 型中断。

#### (2) INT 指令

当执行中断指令  $INT\ n$  时，则产生  $n$  型中断。

#### (3) INTO 指令

若在指令序列执行过程中，上条指令执行的结果使溢出标志位  $OF=1$ ，接着若执行的是 INTO 指令，则引起内部中断，称为 4 型中断；若溢出标志位  $OF=0$ ，该指令将不起作用。

#### (4) 单步执行

当标志位  $T=1$  时, 每执行一条指令, 则引起一次中断, 即指令为单步执行方式, 这种方式常用于程序的调试。单步执行为 1 型中断。

### 2. 硬中断

8088 有两种中断请求线: 非屏蔽中断  $NMI$  线和可屏蔽中断  $INTR$  线, 在这两条线上产生中断请求信号而引起的中断称为硬中断。

#### (1) 可屏蔽中断

出现在  $INTR$  线上的请求信号是电平触发的, 它的出现是异步的, 在 CPU 内部由  $CLK$  的上升沿来同步。在该线上的中断请求信号 (高电平有效) 必须保持到当前指令的结束。

在这条线上出现的中断请求, CPU 是否响应要取决于标志位  $I$  的状态, 若  $I=1$ , 则 CPU 就响应, 可以认为此时 CPU 处于中断状态; 若  $I=0$ , 则 CPU 就不响应。 $I$  位的状态, 可以用指令  $STI$  使其置位 (即开中断); 也可用  $CLI$  指令使其复位 (即关中断)。

要注意: 在系统复位以后, 标志位  $I=0$ ; 此外任一种中断 (内部中断、 $NMI$ 、 $INTR$ ) 被响应后,  $I=0$ 。所以必须在一定的时候用  $STI$  来开放中断。

CPU 是在当前指令周期的最后一个  $T$  状态采样中断请求信号的, 若发现有可屏蔽中断请求, 且中断允许 ( $I=1$ ), 则 CPU 进入中断响应周期。

#### (2) 非屏蔽中断

当  $NMI$  线上出现一个由低向高上跳的高电平中断请求信号后 (持续时间大于两个时钟周期), 不管标志寄存器  $I$  位的状态如何, 当前指令执行完成后, 8088 马上转入中断处理。

此种类型的中断有三种来源: 电源故障, 系统板上随机存储器奇偶校验错, 8087 (协处理器) 中断请求和 I/O 通道检查错。非屏蔽中断的优先权高于可屏蔽中断。

CPU 采样到有非屏蔽中断请求时, 自动给出中断向量号 2 (即 2 型中断), 而不经上述的可屏蔽中断那样的中断响应周期。

### 3. 中断向量表

8088 中有一个简单而又多功能的中断系统。上述的任何一种中断, CPU 响应以后, 都要保护现场 (主要是标志位) 和保护断点 (现行的码段寄存器  $CS$  和指令指针  $IP$ ), 然后转入各自的中断服务程序。在 8088 中各种中断如何转入各自的中断服务程序呢?

8088 在内存的前 1KB (地址为  $00000H \sim 003FFH$ ) 建立了一个中断向量表, 可以容纳 256 个中断向量 (或 256 个中断类型), 每个中断向量占用 4B。在这 4B 中, 包含着这个中断向量 (或这个中断类型) 的服务程序的入口地址——前两个字节为服务程序的  $IP$ , 后两个字节为服务程序的  $CS$ , 如图 7-10 所示。

其中前 5 个中断向量 (或中断类型) 由 Intel 专用, 系统又保留了若干个中断向量, 余下的就可以由用户自己使用, 作为中断源的向量。

外部中断源, 只要在第二个中断响应周期, 向数据总线送出一个字节的中断类型号, 即可以转至相应的中断向量。

### 4. 8088 中的中断响应和处理过程

8088 中各种中断的响应和处理过程是不相同的, 但主要区别在于如何获取相应的中断类型码 (向量号)。



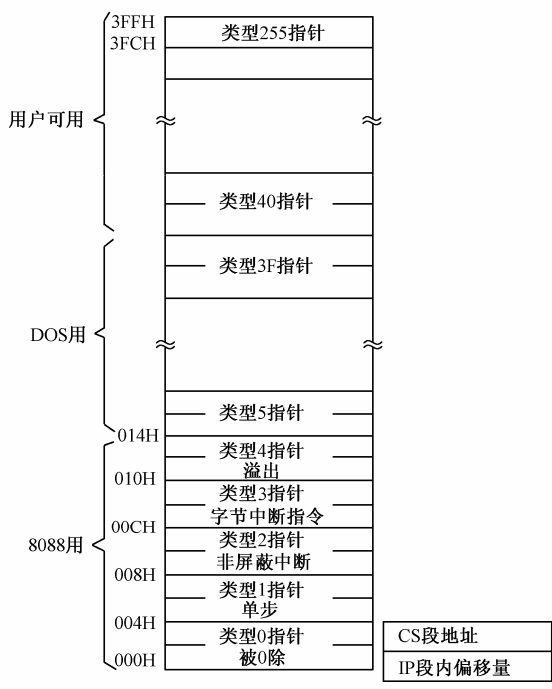


图 7-10 中断向量表

对于硬件（外部）中断，CPU 在当前指令周期的 T 状态采样中断请求信号，如果有可屏蔽中断请求，且 CPU 处在开中断状态（I 标志为 1），则 CPU 转入两个连续的中断响应周期，在第二个中断周期的 T<sub>4</sub> 状态前沿，采样数据线获取由外设输入的类型码；若是采样到非屏蔽中断请求，则 CPU 不经过上述的两个中断响应周期，而在内部自动产生中断类型码 2。

对于软件中断，中断类型码也是自动形成的。

对于 INT n 指令，则类型码即为指令中给定的 n。

8088 在取得了类型码后的处理过程是一样的，其顺序为：

- ① 将类型码乘 4，作为中断向量表的指针；
- ② 把 CPU 的标志寄存器入栈，保护各个标志位，此操作类似于 PUSH F 指令；
- ③ 复制追踪标志 T 的状态，接着清除 I 和 T 标志，屏蔽新的 INTR 中断和单步中断；
- ④ 保存主程序中的断点，即把主程序断点处的 IP 和 CS 的值推入堆栈保护，先推入 CS 的值，再推入 IP 的值；
- ⑤ 从中断向量表中取中断服务程序的入口地址，分别送至 CS 和 IP 中，先取 CS 的值；
- ⑥ 按新地址执行中断服务程序。

在中断服务程序中，通常要保护 CPU 内部寄存器的值（保护现场）及开中断（若允许中断嵌套的话）。在中断服务程序执行完后，要恢复现状，最后执行中断返回指令 IRET。IRET 指令按次序恢复断点处的 IP 和 CS 的值，恢复标志寄存器（相当于 POP F）。于是程序就恢复到断点处继续执行。8088 的中断响应和处理过程可用如图 7-11 所示的流程图来表示。

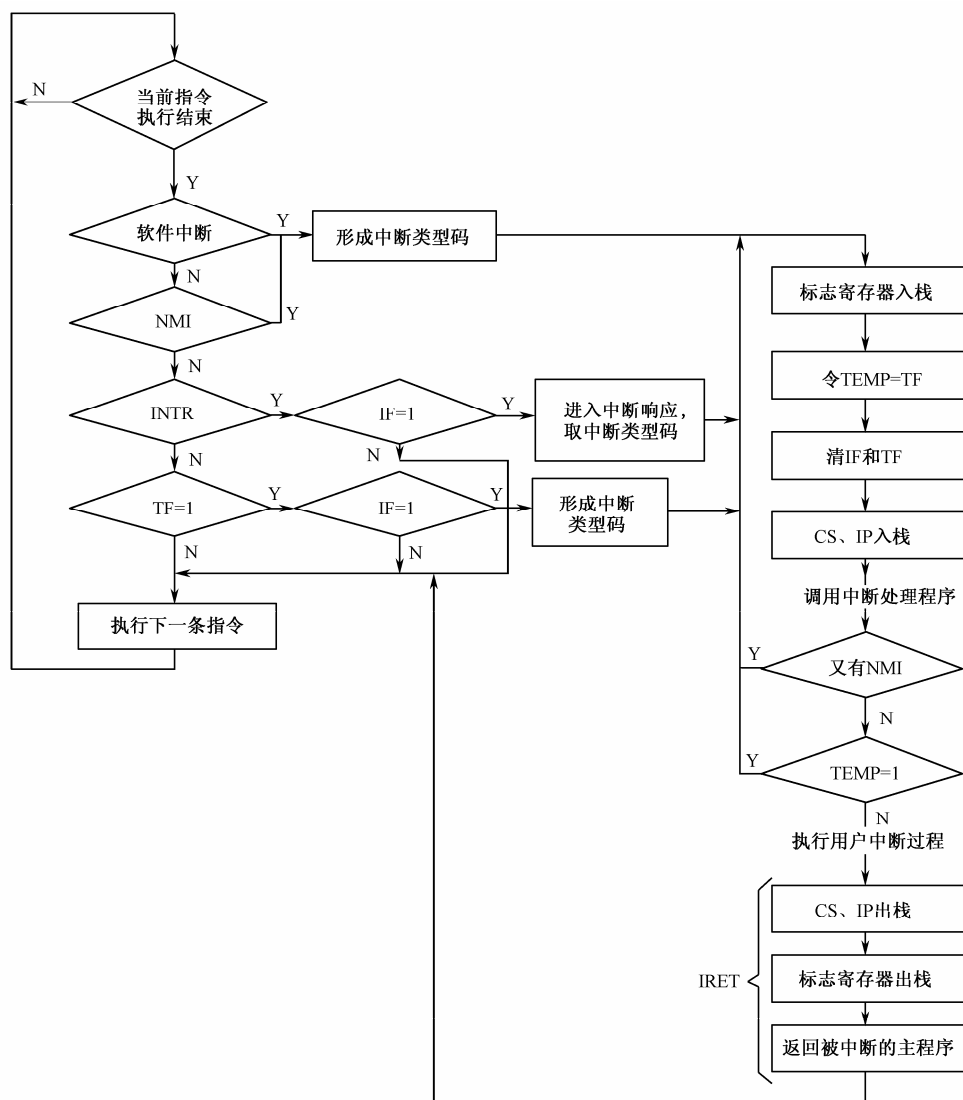


图 7-11 8088 中的中断响应和处理流程图

## 7.5 IBM PC/XT 的中断方式

### 1. IBM PC/XT的中断类型

在 IBM PC/XT 中有三种类型的中断。

#### (1) 内部中断即软中断

包括被 0 除、单步执行、溢出和中断指令（包括断点中断）等。这是由 8088 执行指令产生的中断。

（2）非屏蔽中断NMI

在 IBM PC/XT 中若存储器的读/写奇偶校验错，或者是由 8087（协处理器）的异常所产生的中断，都送至 8088 的 NMI 输入端要求处理。

（3）可屏蔽中断INTR

在 IBM PC/XT 系统中，可能有多个外部设备中断请求信号，而 8088 CPU 的可屏蔽中断输入信号只有一个 INTR。为此，在 IBM PC/XT 系统中，采用中断控制器（8259）将外部设备中断请求信号扩充到 8 个。IBM PC/XT 系统中可屏蔽的中断源及其相应的类型码如表 7-1 所示。

表 7-1 可屏蔽的中断源及其相应的类型码

中断优先级	中 断 源	中断类型码（二进制）
IRQ0	电子钟时间基准	0000 1000
IRQ1	键盘	0000 1001
IRQ2	为用户保留的中断	0000 1010
IRQ3	异步通信（COM2）	0000 1011
IRQ4	异步通信（COM1）	0000 1100
IRQ5	硬盘	0000 1101
IRQ6	软磁盘	0000 1110
IRQ7	并行打印机	0000 1111

2. IBM PC/XT 中系统保留的中断

8088 CPU 最多能处理 256 种不同的中断，其中有 5 个保留为 CPU 专用；又有相当一部分是由磁盘操作系统 DOS 保留为系统用的。所有已经保留的中断类型，用户就不能再使用了，但可使用的仍然有近 200 个中断，这对于绝大部分用户来说已经是足够了。

IBM PC/XT 中保留的中断（所用的 DOS 的版本号不同会有一些不同）中，前 5 个中断类型是 8088 规定的专用中断。BIOS 的中断类型为 0~1F，其功能如表 7-2 所示：

表 7-2 IBM PC/XT 中保留的 BIOS 中断

中断类型号	中 断 功 能	中断类型号	中 断 功 能
0	被 0 除中断	10	CRT 显示 I/O 驱动程序
1	单步中断	11	设备检测
2	NMI	12	存储器容量检测
3	断点中断	13	磁盘 I/O 驱动程序
4	溢出中断	14	RS-232 I/O 驱动程序
5	打印屏幕	15	盒式磁带机处理
6	保留	16	键盘 I/O 驱动程序
7	保留	17	打印机 I/O 驱动程序
8	电子钟定时中断	18	ROM BASIC
9	键盘中断	19	引导（BOOT）
A	保留的硬件中断	1A	一天的时间

续表

中断类型号	中 断 功 能	中断类型号	中 断 功 能
B	异步通信中断 (COM2)	1B	用户键盘 I/O
C	异步通信中断 (COM1)	1C	用户定时器时标
D	硬盘中断	1D	CRT 初始化参数
E	软磁盘中断	1E	磁盘参数
F	并行打印机中断	1F	图形字符集

在这些类型的中断中, 类型号 8~F 就是上述通过 8259 的八级硬件中断; 类型号 5 和 10~1A 是基本外部设备的输入/输出驱动程序和 BIOS 中调用的有关程序; 类型号 1B 和 1C 由用户设定, 1D~1F 指向 3 个数据区域。

中断类型号 20~3F 由 DOS 操作系统使用, 用户程序也可以调用其中的 20~27 号中断。这些中断功能安排如表 7-3 所示。

表 7-3 IBM PC/XT 中保留的 DOS 中断

中断类型号	中 断 功 能	中断类型号	中 断 功 能
20	程序结束	26	磁盘顺序写
21	请求 DOS 功能调用	27	程序结束且驻留内存
22	结束地址	28	DOS 内部使用
23	中止 (Ctrl-Break) 处理	29~2E	DOS 保留使用
24	关键性错误处理	2F	DOS 保留使用
25	磁盘顺序读	30~3F	DOS 保留使用

40 号以后的中断类型可由用户程序安排使用。

## 7.6 中断控制器 8259A

在中断控制过程中, 中断源的识别和优先权的确定可以用硬件排队电路等实现, Intel 8259A 可编程中断控制器就是为完成这些任务而设计的一种器件。它不是 I/O 接口, 而是一种中断管理芯片, 统称 PIC (Programmable Interrupt Controller)。8259A 可编程中断控制器用来管理 8 级优先中断, 并可多个 8259A 级联起来, 构成 64 级中断优先级管理系统, 而无需外加电路; 它具有多种工作方式, CPU 可以通过编程设定或改变它的工作方式; CPU 响应中断时, 8259A 能自动提供中断入口地址, 而使 CPU 转向相应的中断处理程序。中断入口地址可以由用户设定, 且入口地址可以选定在任何存储单元。

8259A 的主要功能为:

- ① 具有 8 级优先权控制, 通过芯片级联可扩展至 64 级优先权控制。
- ② 每一级中断均可通过编程屏蔽或允许。
- ③ 在中断响应周期可提供相应的中断类型号。
- ④ 有多种工作方式, 可通过编程选择。
- ⑤ 可与 CPU 直接连接, 不需外加硬件电路。

### 7.6.1 8259A的内部结构

8259A 采用 NMOS 工艺，只需要单一的+5V 电源，它的内部电路为静态电路，因此不需要时钟输入，其内部结构如图 7-12 所示。

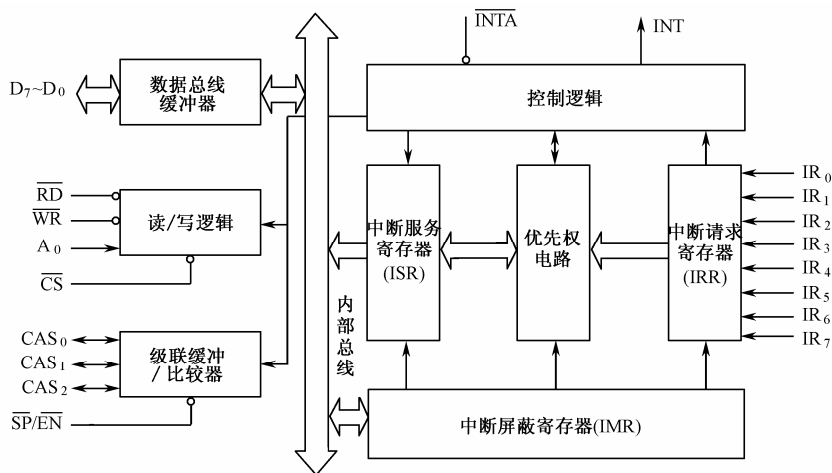


图 7-12 8259A 内部结构

#### 1. 数据总线缓冲器

数据总线缓冲器是三态、双向、8 位的缓冲器， $D_7 \sim D_0$  用于和 CPU 的数据总线连接，CPU 通过数据总线缓冲器向 8259A 传送命令码，或从 8259A 读取状态字。在中断响应时，8259A 通过数据总线缓冲器向 CPU 提供 CALL 指令的操作码和调用子程序入口地址的低 8 位和高 8 位，以及中断类型码。

#### 2. 中断请求寄存器 (IRR)

中断请求寄存器用来寄存所有 IR 输入线输入的中断请求信号，即保存正在请求服务的中断级。有一个请求输入线就有一个触发器来保存相应的状态，共有  $IR_0 \sim IR_7$  8 条输入线，连接 8 个 I/O 设备的中断请求信号。当  $IR_0 \sim IR_7$  中的某一条请求线上升为高电平时，IRR 中的相应位置“1”。

#### 3. 优先权电路

优先权电路的主要作用是确定中断请求寄存器 IRR 中各位的优先级，并确定能否向 CPU 申请中断。当 IRR 中有中断请求触发器置位时，优先权电路就选出其中的未被屏蔽的最高优先级，并对该优先级编码；然后再按照一定的优先级方式同中断服务寄存器 (ISR) 中的最高优先级相比较，以便确定最终有无向 CPU 申请中断的资格。若可以申请中断，则 CPU 响应中断请求；发来第一个  $\overline{INTA}$  脉冲时，将级别最高者放入中断服务寄存器 (ISR) 中的相应位置。

#### 4. 中断服务寄存器 (ISR)

中断服务寄存器 (ISR) 的主要作用是保存当前被 CPU 服务的中断级，也就是记录正在被处理的中断请求。当某一级中断被 CPU 响应，并执行它的中断服务程序时，中断服务

寄存器 **ISR** 中的相应位置“1”，并一直保持该状态（非自动结束中断方式），直到 CPU 发出结束中断命令 **EOI** 为止。在中断嵌套情况下，**ISR** 中会有多个位被置“1”。在 CPU 响应中断，发出第一个  $\overline{\text{INTA}}$  时，最高优先级的状态在 **ISR** 中相应位置“1”，同时该位的编码写入中断向量寄存器的低 3 位（高 5 位由初始化编程时写入），以备在第二个  $\overline{\text{INTA}}$  到来时送出对应的中断类型号。

### 5. 中断屏蔽寄存器（IMR）

中断屏蔽寄存器的主要作用是对各中断源的中断请求信号（ $\text{IR}_0 \sim \text{IR}_7$ ）实现开关控制。这个寄存器中保存对输入请求线上的屏蔽信息，这些信息是由 CPU 送给 8259A 的操作命令  $\text{OCW}_1$  来设定的。当某位为“1”时，表示禁止相应的中断请求进入优先权电路。

### 6. 控制逻辑

控制逻辑内部包括内部控制电路、中断控制电路、初始化命令寄存器组和操作命令寄存器组，它根据 CPU 对 8259A 编程设定的工作方式产生 8259A 内部控制信号，并根据中断请求寄存器（**IRR**）和优先权电路的判别结果，在适当时候向 CPU 发生中断请求信号，并接收 CPU 发来的中断响应信号  $\overline{\text{INTA}}$ ，控制提供中断类型号。

### 7. 读/写逻辑

接收 CPU 来的控制信号，包括端口控制信号  $\text{A}_0$  和  $\overline{\text{CS}}$ 、数据方向控制信号  $\overline{\text{RD}}$  和  $\overline{\text{WR}}$ 。控制将 CPU 送来的初始化命令 **ICW** 和操作命令 **OCW** 存入 8259A 内部相应的寄存器，用以规定 8259A 的工作方式。

### 8. 级联缓冲/比较器

一片 8259A 最多可构成八级中断（ $\text{IR}_0 \sim \text{IR}_7$ ），要想扩展中断源，必须多片连在一起，即采用级联方式。级联缓冲/比较器的功能有两个，一是提供级联控制，二是提供缓冲控制。 $\text{CAS}_2 \sim \text{CAS}_0$  用于提供级联信号，主片输入，从片输出。

对于 8088 CPU 而言，8259A 的工作过程是：

① 某一条或某几条中断请求线（ $\text{IR}_0 \sim \text{IR}_7$ ）有中断申请，变为高电平，使中断请求寄存器（**IRR**）的相应位置“1”。

② **IMR** 对 **IRR** 屏蔽。未被屏蔽的请求信号经优先权电路判别最高优先级，再经由优先权方式确定没有更高级优先权的中断，则 8259A 的 **INT** 端输出为“1”，向 CPU 提出中断请求。

③ CPU 响应中断后发出中断响应信号。在中断响应过程中，CPU 要发出两次  $\overline{\text{INTA}}$  信号。当 8259A 收到第一个  $\overline{\text{INTA}}$  信号后，**ISR** 中当前被选中的最高优先级对应的那一位置“1”，同时 **IRR** 中的相应位被清“0”，表示该位上的中断请求已被 CPU 所接受。

④ 8259A 收到第二个  $\overline{\text{INTA}}$  信号后，驱动数据总线将对应的中断类型码输出。

⑤ 如果是自动结束中断方式（**AEIOI**），则在第二个  $\overline{\text{INTA}}$  脉冲结束时将 **ISR** 中置“1”的位复位，否则该位的“1”将一直保持，直到 CPU 发出 **EOI** 命令为止。

## 7.6.2 8259A的引脚功能

8259A 是一个 28 引脚的双列直插式芯片，其引脚信号可参见图 7-12。

①  $D_7 \sim D_0$ : 双向三态数据总线, 8 位, 用于传送控制和状态信息、中断向量、中断类型码等。

②  $IR_7 \sim IR_0$ : 中断请求输入信号, 8 级中断源。一般情况下,  $IR_7$  的中断优先级别最低, 而  $IR_0$  的中断优先级别最高。能够使  $IRR$  的相应位置“1”的  $IR_i$  的有效状态有两种, 可用命令来设定。其一是边沿触发, 即  $IR$  必须由低变高产生一个跳变才能使  $IRR$  相应位的触发器置“1”, 提出中断请求; 其二是电平触发, 即只要  $IR$  是一个高电平即可。边沿触发具有锁存中断源中断请求信号的能力, 只要有一次跳变,  $IRR$  就能锁存中断请求状态, 而与中断源无关。而在电平触发方式中,  $IRR$  不能锁存中断源的中断请求状态, 即在中断响应之前, 中断源必须有能力保持中断请求信号为高电平, 否则将视为自行撤销中断请求。

③  $\overline{RD}$ : 读信号, 输入。当  $\overline{RD}=0$  时, 8259A 将状态信息送至数据总线供 CPU 使用。

④  $\overline{WR}$ : 写信号, 输入。当  $\overline{WR}=0$  时, 8259A 接收数据总线上 CPU 传来的数据。

⑤  $\overline{CS}$ : 片选信号, 输入, 低电平有效。当  $\overline{CS}=0$  时 8259A 被选中, 允许 CPU 对 8259A 进行读/写操作;  $\overline{CS}=1$  时芯片未被选中。

⑥  $A_0$ : 地址线, 输入。该信号与  $\overline{CS}$ 、 $\overline{RD}$ 、 $\overline{WR}$  一起用来选择 8259A 的内部寄存器。该信号通常直接连到地址总线的  $A_0$ , 8259A 用该信号控制接收 CPU 的命令字或向 CPU 发状态信息。8259A 有 4 个初始化命令字和 3 个操作命令字, 共需要 7 个寄存器, 但却只有一条地址线  $A_0$ , 只能有 2 个端口地址, 即一个奇地址和一个偶地址, 所以 7 个寄存器只好公用 2 个地址, 并以读/写顺序和命令字特征码加以区别。

⑦  $INT$ : 中断请求信号, 输出。只要 8259A 的中断逻辑判定中断请求信号有效, 就在这个引脚上产生一个高电平, 可接到 CPU 的中断输入端。

⑧  $\overline{INTA}$ : 中断响应信号, 输入, 是来自于 CPU 的响应脉冲, 8259A 根据  $ISR$  中的置位情况提供相应的中断类型码。8259A 在第二个  $\overline{INTA}$  时向 CPU 提供的中断类型码的高 5 位是用户在程序中规定的, 而低 3 位则是由 8259A 自动产生的。

⑨  $V_{CC}$ ,  $GND$ : 电源 (+5V) 和地。

⑩  $CAS_0 \sim CAS_2$ : 级联信号, 双向, 形成 8259A 的专用总线, 以便构成多片 8259A 的级联结构。当 8259A 是主片时,  $CAS_0 \sim CAS_2$  是输出线, 在 CPU 响应中断时, 输出被选中的从片代码。当 8259A 是从片时,  $CAS_0 \sim CAS_2$  是输入线, 在 CPU 响应中断时, 接收主片送出的被选中的从片代码, 然后在从片内将接收来的代码与本从片代码相比较, 看是否一致, 从而确定 CPU 响应的是不是本从片的中断请求。

⑪  $\overline{SP}/\overline{EN}$ : 从片编程/允许缓冲器信号, 双向, 低电平有效、双重功能引脚。当工作在缓冲器方式时, 它是输出信号, 用做允许缓冲器接收和发送的控制信号 ( $\overline{EN}$ ), 即启动 8259A 至 CPU 之间的数据总线缓冲器。如图 7-13 所示, 8259A 通过总线缓冲器 8286 (或者用 74245) 与系统数据总线连接, 控制 8286 的传送方向。当工作在非缓冲器方式时, 它是输入信号, 用来指明系统中的 8259A 究竟是作为主片工作 ( $\overline{SP}/\overline{EN}=1$ ) 的, 还是作为从片工作 ( $\overline{SP}/\overline{EN}=0$ ) 的, 要由程序命令来设定。缓冲器方式适合于 8259A 的级联。

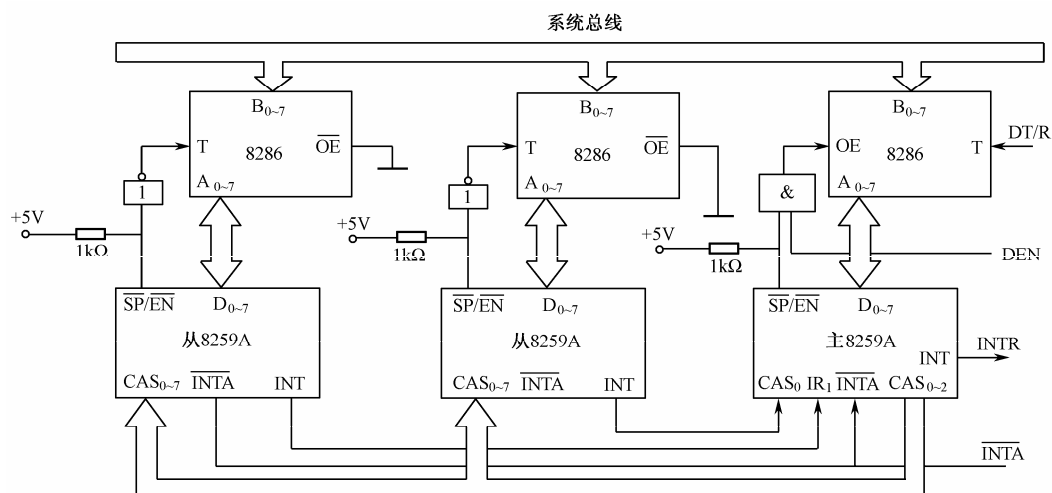


图 7-13 缓冲方式原理图

### 7.6.3 8259A的工作方式

中断控制器 8259A 有四种主要的工作方式：全嵌套、循环优先级、特定屏蔽和程序查询方式；还有四种从属的工作方式：中断结束方式、读状态、中断请求触发方式和缓冲器方式。另外，它还被分为两大工作类型，即单片工作和多片级联工作。

8259A 是可编程芯片，可以通过程序命令来确定 8259A 的工作方式。8259A 有两种命令，一种是初始化命令，另一种是操作命令，它们用于控制 8259A 的中断管理。

下面分别介绍上述的工作方式。

### 1. 全嵌套方式

这是一种最普通的工作方式。8259A 在初始化工作完成后若未设定其他的工作方式，就自动进入全嵌套方式。这种方式的特点是：

- ① 中断请求的优先级固定，其顺序是  $\text{IR}_0$  最高，依次降低， $\text{IR}_7$  最低。
- ② 中断服务寄存器（ISR）保存优先权电路确定的优先级状态，相应位置“1”，并且一直保持这个服务“记录”状态，直到 CPU 发出中断结束命令为止。
- ③ 在 ISR 置位期间，不再响应同级及较低级的中断请求，而高级的中断请求如果 CPU 开放中断的话仍能够得到中断服务。

④  $IR_7 \sim IR_0$  的中断请求输入可分别由中断屏蔽寄存器 (IMR) 的相应位 ( $D_7 \sim D_0$ ) 屏蔽与允许, 对某一位的屏蔽与允许操作不影响对其他位的中断请求操作。

全嵌套工作方式由 ICW<sub>4</sub> 的 D<sub>4</sub>=0 来确定。

## 2. 循环优先级方式

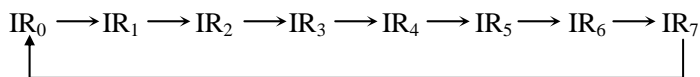
循环优先级方式是 8259A 管理优先级相同的设备时所采用的中断管理方式，它包括自动循环优先级方式和特殊循环优先级方式。

(1) 自动循环

日 突 出 地 点 坡 顶 附 近



谓各设备优先级相同，是指它们的地位相同，受服务的机会均等，但是毕竟各中断源的优先级需要排出一个顺序，否则同时有多个中断源申请中断时计算机无法处理，于是排出的优先级由高到低的顺序为：



这是一个循环套，有一个最低优先权指针，哪一台设备刚被服务后，它就被赋予最低优先权指针。例如， $IR_7$  刚被服务，它就被赋予最低优先权指针，按照循环顺序， $IR_0$  的优先级就是最高优先级；如果  $IR_4$  刚被服务， $IR_4$  就被赋予最低优先权指针，按照优先级循环顺序， $IR_5$  的优先级就最高。这样，当一台设备提出中断请求后，在最不利的情况下（此时它的优先级最低），待其他 7 台设备被轮流服务一次以后，它变为最高优先级，从而得到系统的服务。但是如果不是在循环优先级方式下（包括自动循环和特殊循环）工作，它可能永远得不到系统的服务。

自动循环优先级方式由  $OCW_2$  的  $R=1$ 、 $SL=0$  来确定。

## （2）特殊循环

特殊循环优先级方式与自动循环优先级方式的不同之处在于：在自动循环优先级方式中，某一设备在被服务之后被确定为最低优先级；而在特殊循环优先级方式中，通过编程来确定某一设备为最低优先级。如  $IR_5$  被指定为最低优先级，则  $IR_6$  的优先级最高。

特殊循环优先级方式由  $OCW_2$  的  $R=1$ 、 $SL=1$  来确定， $L_2L_1L_0$  用于指定最低优先级的二进制编码。一般来说，在命令控制字中，凡是采用“ $L_2L_1L_0$ ”的都有“特殊”的含义。

## 3. 特定屏蔽方式

8259A 的每个中断请求输入信号都可由中断屏蔽寄存器（IMR）的相应位进行屏蔽，IMR 的  $D_0$  对应  $IR_0$ ， $D_1$  对应  $IR_1$ …… $D_7$  对应  $IR_7$ 。相应位为“1”则屏蔽中断输入，相应位为“0”则允许中断输入。IMR 寄存器由操作命令  $OCW_1$  进行设置。对中断请求输入信号的屏蔽方式一般有两种：正常屏蔽方式和特定屏蔽方式。

在正常屏蔽方式中，每一个屏蔽位对应一个中断请求输入信号，屏蔽某一个中断请求输入信号对其他请求信号没有影响，未被屏蔽的中断请求输入信号仍然按照设定的优先级顺序进行工作，而且保证当某一级中断请求被响应服务时，同级和低级的中断请求将被禁止。如果 CPU 允许中断，则高级的中断请求还会被响应，实现中断嵌套。

特定屏蔽方式也叫特殊的中断屏蔽方式，当设定了特定屏蔽方式后，IMR 中为“1”的位仍要屏蔽相应的中断请求输入信号，但所有未被屏蔽的位被全部开放，无论优先级是低还是高，都可以申请中断，并且都可能得到 CPU 的响应并为之服务。也就是说，这种方式抛弃了同级或低级中断被禁止的原则，任何级别的未被屏蔽的中断请求都会得到响应，所以，可以有选择地设定 IMR 的状态，开启需要的中断输入。

特定屏蔽方式由  $OCW_3$  的  $ESMM$  和  $SMM$  确定，设定时  $ESMM=1$ 、 $SMM=1$ ，复位时  $ESMM=1$ 、 $SMM=0$ 。

## 4. 程序查询方式

程序查询方式不使用中断，是用软件寻找中断源并为之服务的工作方式。在这种方式

下, 8259A 不向 CPU 发送 INT 信号 (实际上是 8259A 的 INT 信号不连到 CPU 的 INTR 信号上), 或者 CPU 关闭自己的中断允许触发器, 使  $IF=0$ , 禁止中断输入。申请中断的优先级不是由 8259A 提供的中断类型码而是由 CPU 发出查询命令得到的。

查询时, CPU 先向 8259A 发出查询命令, 8259A 接到查询命令后, 就把下一个 IN 指令 (对偶地址端口的读指令) 产生的  $\overline{RD}$  脉冲作为中断响应信号, 此时, 若有中断请求信号, 则在 ISR 中相应位置 “1”, 并把该优先级送至数据总线。在  $\overline{RD}$  期间 8259A 送至数据总线供 CPU 读取查询的代码格式为:

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
I	—	—	—	—	W <sub>2</sub>	W <sub>1</sub>	W <sub>0</sub>

其中, I 是中断请求标志,  $I=1$  表示有中断请求, 此时  $W_2W_1W_0$  有效,  $W_2W_1W_0$  就表示申请服务的最高中断优先级。 $I=0$  表示没有中断请求, 此时  $W_2W_1W_0$  无效。例如读入的查询代码是 83H, 则表示有中断请求, 申请中断的优先级输入是  $IR_3$ 。

在查询方式下, CPU 不需执行中断响应周期, 不必安排中断向量表, 8259A 能自动提供最高优先级中断请求信号的二进制代码, 供 CPU 查询。该方式使用方便, 可扩充中断优先级数目, 扩充数目超过 64 级以上 (此时不是中断级联方式, 而是一般的端口连接。在查询时, 只涉及 8259A 端口地址。显然, 在查询方式下, 能够扩展的 8259A 的数目仅限于系统的 I/O 空间容量)。

查询方式是由  $OCW_3$  的  $P=1$  来确定的。

### 5. 中断结束方式

所谓中断结束方式是指中断如何结束的方法, 这里的 “结束” 不是指中断服务程序的结束。中断服务程序的结束用 IRET 指令就可完成, 这里的 “结束” 是指如何和何时使 8259A 中的 ISR 中的相应位清 0。ISR 中某位为 “1”, 表示 CPU 正在为之服务; 某位为 “0” 表示 CPU 已经停止 (结束) 为之服务。而 IRET 指令主要是恢复程序的断点, 它并不能使 ISR 的相应位清 0。

8259A 的中断结束方式有两种: 命令中断结束方式 (EOI) 和自动中断结束方式 (AEOI)。

#### (1) 自动结束

在自动中断结束 (AEOI) 方式下, 8259A 自动地在最后一个  $\overline{INTA}$  中断响应脉冲的后沿将中断服务寄存器 ISR 中的相应位清 0。这种方式的过程是: 中断请求, CPU 响应, 发第一个  $\overline{INTA}$ , ISR 相应位置 “1”, CPU 发第二个  $\overline{INTA}$ , 8259A 提供中断类型码, ISR 相应位清 0, 结束。显然, ISR 的相应置 “1” 位在 CPU 中断响应周期内自生自灭, 因此在 ISR 中不会有两个或两个以上的置 “1” 位。

自动中断结束方式 (AEOI) 的应用场合一般是, 8259A 单片系统, 或不需要嵌套的多级中断系统。AEOI 方式只能用于主片 8259A, 不能用于从片 8259A。

自动中断结束方式由  $ICW_4$  的  $AEOI=1$  确定。

#### (2) 命令结束

命令中断结束方式 (EOI) 是在中断服务程序返回之前, 向 8259A 发中断结束命令

(EOI), 使 ISR 中的相应位清 0。它包括两种情况:

① 非特殊 EOI 命令: 全嵌套方式下的中断结束命令称为非特殊 EOI 命令, 该命令能自动地把当前 ISR 中的最高优先级的那一位清 0。

非特殊 EOI 命令是由 OCW<sub>2</sub> 的 R=0、SL=0、EOI=1 确定的。

② 特殊 EOI 命令: 非全嵌套方式下的中断结束命令称为特殊 EOI 命令。在非全嵌套方式下, 由于无法确定最后响应的是哪一级中断 (非全嵌套方式的各中断源没有固定的优先级别, 因此也就不知道谁高谁低), 所以应向 8259A 发出特殊 EOI 命令, 即指定哪一级中断返回, 使其 ISR 中的相应位清 0。

特殊 EOI 命令是由 OCW<sub>2</sub> 的 R=0、SL=1、EOI=1 确定的, 由 L<sub>2</sub>L<sub>1</sub>L<sub>0</sub> 指定 ISR 中要复位的相应位的二进制编码。

## 6. 读 8259A 状态

读 8259A 的状态是指读 8259A 内部的 IRR、ISR 和 IMR 的内容。

① 读 IRR: 先发出 OCW<sub>3</sub> 命令 (使 RR=1, RIS=0, 地址 A<sub>0</sub>=0), 在下一个  $\overline{RD}$  脉冲到来时可读出 IRR, 其中包含尚未被响应的中断源情况。

② 读 ISR: 先发出 OCW<sub>3</sub> 命令 (使 RR=1, RIS=1, 地址 A<sub>0</sub>=0), 在下一个  $\overline{RD}$  脉冲到来时可读出 ISR, 其中包含正在服务的中断源情况, 从中也可看出中断嵌套情况。

③ 读 IMR: 不必先发 OCW<sub>3</sub>, 只要读出奇地址端口 (A<sub>0</sub>=1), 即可读出 IMR, 其中包含设置的中断屏蔽情况。

## 7. 中断请求触发方式

8259A 的中断请求寄存器 IRR 中有 8 个中断请求触发器, 分别对应 8 个中断请求信号的输入端 IR<sub>0</sub>~IR<sub>7</sub>。这些触发器的触发方式有两种, 即边沿触发和电平触发。

### (1) 边沿触发

当输入端有从低电平到高电平的正跳变时, 则产生中断请求 (IRR 中相应位的触发器被触发置“1”, 而不是直接向 CPU 申请中断)。此后, 即使输入端仍然保持高电平也不会再产生中断。也就是说, 只有正跳沿才能产生中断。

边沿触发方式由 ICW<sub>1</sub> 的 LTIM=0 确定。

### (2) 电平触发

当输入端为高电平时产生中断请求 (只要为高电平就可以, 不需要脉冲跳变)。但需要注意的是, 在电平触发方式下, 在发出 EOI 命令以前, 或 CPU 开放中断以前, 必须去掉中断请求信号 (使其变为低电平), 否则将产生第二次中断。

电平触发方式由 ICW<sub>1</sub> 的 LTIM=1 确定。

## 8. 缓冲器方式

所谓缓冲器方式就是在 8259A 和数据总线之间挂接总线驱动器的方式。在缓冲器方式下,  $\overline{SP}/\overline{EN}$  引脚将使用  $\overline{EN}$  功能, 并使之输出一个有效低电平, 开启缓冲器工作。该方式多用于级联的大系统中。

缓冲器方式由 ICW<sub>4</sub> 的 BUF=1 确定。

## 9. 特殊的全嵌套方式

该方式适用于多片级联, 且必须将优先级保存在各从片 8259A 中的大系统。该方式与

普通的全嵌套方式的工作情况基本相同，有两点区别：

① 当某从片的一个中断请求被 CPU 响应后，该从片的中断仍未被禁止（即没有被屏蔽），即该从片中的高级中断仍可提出申请。（全嵌套方式中这样的中断是被屏蔽的，因为这种中断对从片而言后者是高级中断，可以嵌套，但对主片而言，由于它们来自于同一个从片，故中断优先级相同，而在全嵌套方式中，同级和低级中断是被禁止的。）

② 在某个中断源退出中断服务程序之前，CPU 要用软件检查它是否是这个从片中的唯一中断。检查的办法是：送一个非特殊的中断结束命令（EOI）给这个从片，然后读它的 ISR，检查是否为 0，若为 0 则唯一，即只有这一个中断在被服务，没有嵌套；若不为 0 则不唯一，说明还有其他的中断在被服务，该中断是嵌套在其他中断里的。只有唯一时，才能把另一个非特殊 EOI 命令送至主片，结束此从片的中断。否则，如果过早地结束主片的工作记载而从片尚有未处理完的嵌套中断的话，整个系统的中断嵌套环境就会混乱。

特殊的全嵌套方式由 ICW<sub>4</sub> 的 SFNM=1 确定。

### 10. 多片级联方式

在级联系统中，每个从片的中断请求输出线 INT 直接连到主片的某个中断请求输入线上，主片的 CAS<sub>0</sub>~CAS<sub>2</sub> 是输出线，输出被响应的从片代码，从片的 CAS<sub>0</sub>~CAS<sub>2</sub> 是输入线，接收主片发出的从片代码，以便与自身代码相比较。级联方式的要点如下：

① 一个 8259A 主片至多带 8 个从片，可扩展至 64 级。

② 缓冲方式下，主片和从片的设定由 ICW<sub>4</sub> 的 M/S 位确定，M/S=1 是主片，M/S=0 是从片。M/S 的状态在 BUF=1 时有意义。

③ 在非缓冲方式下，主片和从片由  $\overline{SP}/\overline{EN}$  引脚的  $\overline{SP}$  功能确定， $\overline{SP}=1$  是主片， $\overline{SP}=0$  是从片。

④ 在级联系统中，主片的三条级联线相当于从片的片选信号，从片的 INT 是主片的中断请求输入信号。

⑤ 主片和从片需要分别进行初始化操作，可设定为不同的工作方式。

级联方式由 ICW<sub>1</sub> 的 SNGL=0 确定。

上述的各种工作方式中，全嵌套方式、自动中断结束方式、中断请求触发方式、缓冲器方式、特殊的全嵌套方式、级联方式等是由初始化命令字 ICW 来设定的，而循环优先级方式、特定屏蔽方式、查询方式、命令中断结束方式、读 8259A 状态等是由操作命令字 OCW 来设定的。

### 7.6.4 8259A 的编程

8259A 是一个可编程器件。为了使 8259A 实现预定的中断管理功能，并按预定的方式工作，就必须对它进行初始化编程。所谓初始化编程是指系统在上电或复位后对可编程器件进行控制字设定的一段程序。8259A 的命令控制字包括两个部分，即初始化命令字和操作命令字。初始化命令字一般在系统复位后的初始化编程中设置，用于确定 8259A 的基本工作方式，设置以后一般保持不变。操作命令是在初始化以后的正常工作中写入的，它实现对 8259A 的状态、中断方式和过程的动态控制，在工作中可随时写入操作命令字以修改某些控制方式。

8259A 内部有 7 个寄存器，分为两组：初始化命令寄存器组和操作命令寄存器组。初始化命令寄存器组包括 4 个寄存器：ICW<sub>1</sub>~ICW<sub>4</sub> 对应的寄存器。操作命令寄存器组包括 3 个寄存器：OCW<sub>1</sub>~OCW<sub>3</sub> 对应的寄存器。

由于 8259A 只有一条地址线 A<sub>0</sub>，所以它只能有两个端口地址；而 8259A 有 7 个命令字，每个命令字要写入相应的寄存器。为此，采取以下几点措施：第一，以端口地址区分；第二，把命令字中的某些位作为特征码来区分；第三，以命令字的写入顺序来区分。

在 PC/XT 中，8259A 的两个端口地址分别为 20H 和 21H。

下面具体讨论 8259A 的命令字。

### 1. 初始化命令字

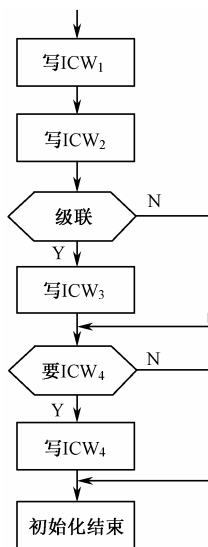


图 7-14 8259A 的 ICW 写入顺序

初始化命令字有 4 个：ICW<sub>1</sub>~ICW<sub>4</sub>。8259A 在进入正常工作之前，必须将系统中的每一个 8259A 进行初始化设置，以此建立 8259A 的基本工作条件。写入的初始化命令字一般为 2~4 个（在某些条件下，4 个初始化命令字并非必须全部写入），最多为 4 个，然而，ICW<sub>1</sub> 使用偶地址，而 ICW<sub>2</sub>、ICW<sub>3</sub>、ICW<sub>4</sub> 都使用奇地址。为了相互区别，初始化命令字的写入必须有一个固定的顺序，其顺序如图 7-14 所示。

系统上电或复位以后，对 8259A 第一件要做的工作就是按图 7-14 的顺序写入初始化命令字。初始化命令字格式如图 7-15 所示。

初始化命令字 ICW<sub>1</sub> 的主要功能是：确定级联方式，触发方式。

写入 ICW<sub>1</sub> 后，8259A 内部自动复位，其复位功能为：

① 初始化命令字顺序逻辑重新置位，准备接收 ICW<sub>2</sub>~ICW<sub>4</sub>。

② 清除 IMR 和 ISR。

③ IRR 状态可读。

④ 优先级排队，IR<sub>0</sub> 最高，IR<sub>7</sub> 最低。

⑤ 特殊屏蔽方式复位。

⑥ 设定中断请求信号由低变高的边沿触发有效。

⑦ 自动 EOI 循环方式复位。

初始化命令字 ICW<sub>2</sub> 的主要功能是：确定中断向量，中断类型码。

初始化命令字 ICW<sub>3</sub> 的主要功能是：确定主片/从片的级联状态，即确定主片的连接位和从片的编码。

初始化命令字 ICW<sub>4</sub> 的主要功能是：选择 CPU 系统，确定中断结束方式，规定是主片还是从片，选择是否采用缓冲方式。

初始化命令字一定要在系统复位后首先写入 8259A，写入时要严格按照图 7-14 的顺序，不允许颠倒。

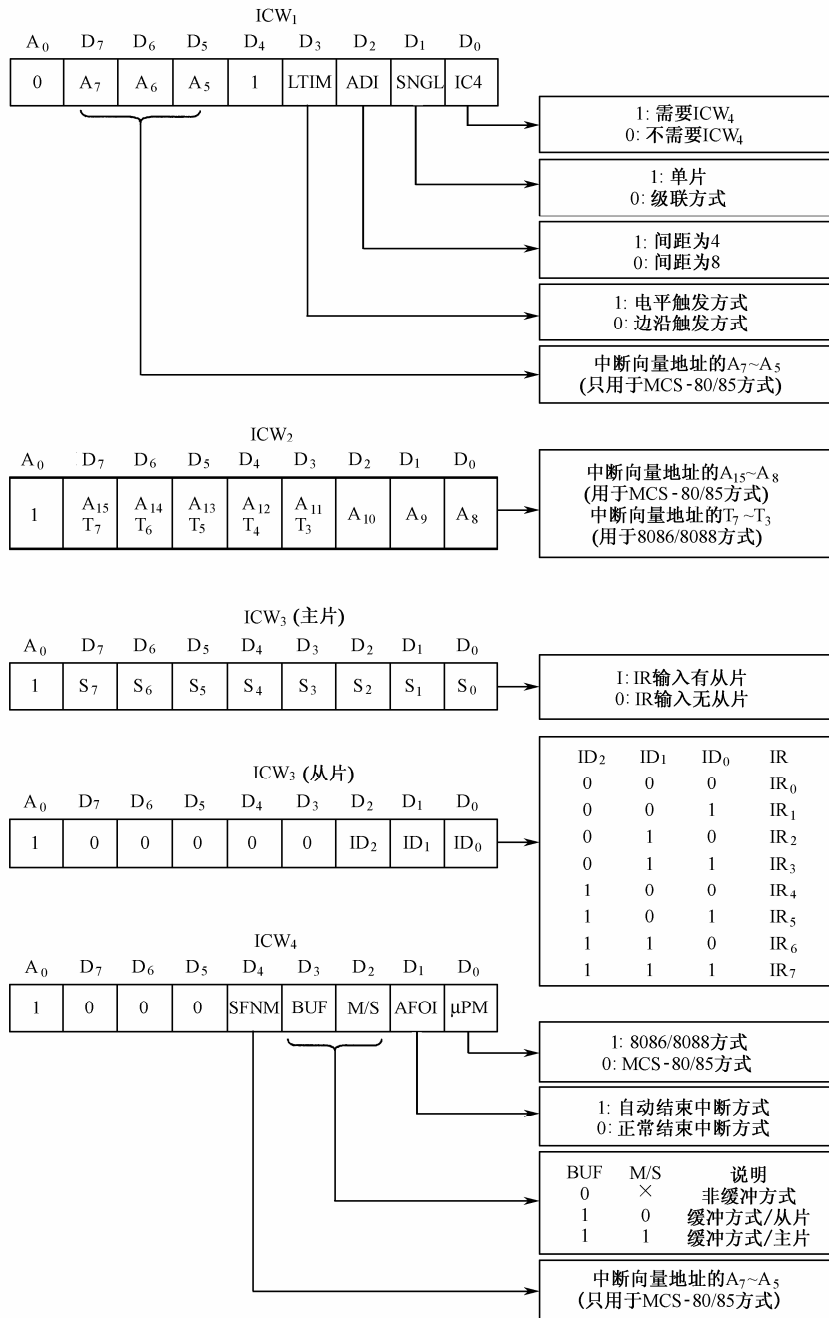


图 7-15 8259A 初始化命令字格式

写完初始化命令字后，8259A 已经建立了基本的工作环境，可以接受中断请求，也可以写入操作命令字 **OCW** 来改变某些中断管理方式。操作命令字可以随时写入、修改，但初始化命令字一经写入一般不再改动。如果在写入初始化命令字后不写入操作命令字，则 8259A 便处于全嵌套工作方式，即中断优先级为 **IR<sub>0</sub>** 最高，**IR<sub>7</sub>** 最低，禁止同级及低级中断，高级

中断可嵌套处理。

## 2. 操作命令字

在初始化命令字写入 8259A 之后, 8259A 就准备接收中断请求输入信号了。在 8259A 工作期间, CPU 可以随时通过操作命令字使 8259A 完成各种不同的工作方式。8259A 有三种操作命令字:  $OCW_1$ 、 $OCW_2$  和  $OCW_3$ , 在写入时, 它们与初始化命令字不同, 它们不是按一定的顺序写入, 而是按设计者的要求写入。操作命令字如图 7-16 所示。

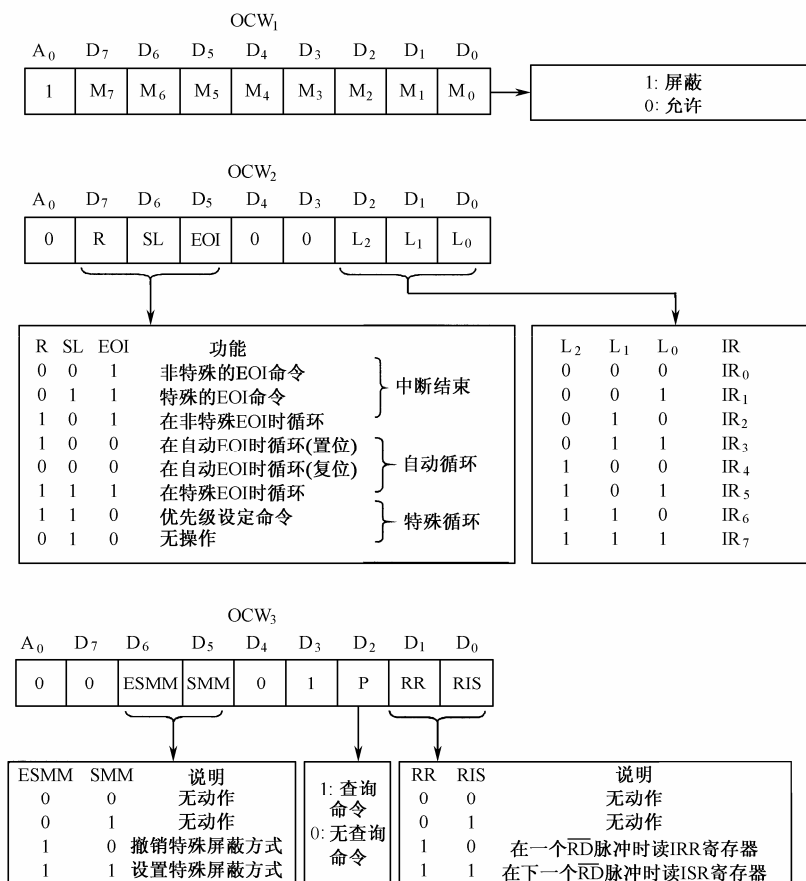


图 7-16 8259A 操作命令字格式

操作命令字  $OCW_1$  的主要功能是: 保存中断屏蔽字。

操作命令字  $OCW_2$  的主要功能是: 控制 8259A 的中断循环优先级方式及发送命令中断结束方式。

操作命令字  $OCW_3$  的主要功能是: 设定查询方式和特殊屏蔽方式。

【例 7-1】8259A 单片应用。在某 8088 系统中扩展一片中断控制器 8259A, 其端口地址由 74LS138 译码器译码选择, 假设为 8CH 和 8DH。中断源的中断请求线连到  $IR_7$  输入线上, 边沿触发方式,  $IR_7$  的中断类型码为 77H, 其他条件保持 8259A 的复位设置状态。要求: 写出 8259A 的初始化程序, 中断类型码为 77H 的中断向量设置程序。

(1) 8259A 的初始化程序。

初始化程序包括写入 ICW<sub>1</sub>、ICW<sub>2</sub> 和 ICW<sub>4</sub> (由于单片使用, 不需写入 ICW<sub>3</sub>), 并且必须按规定的顺序写入。

① ICW<sub>1</sub> 命令字。单片, 边沿触发, 需要 ICW<sub>4</sub>, 故为 00010011B=13H, 写入偶地址。

② ICW<sub>2</sub> 命令字。IR<sub>7</sub> 的中断类型码为 77H, 即可作为 ICW<sub>2</sub> 命令字写入, 写入奇地址。

③ ICW<sub>4</sub> 命令字。8088 CPU 一般为全嵌套方式, 正常 EOI 结束, 非缓冲方式, 故命令字的组合为 00000001B=01H, 写入奇地址。

④ OCW<sub>1</sub> 命令字。系统只使用了 IR<sub>7</sub>, 为防止干扰, 产生误动作, 应将 IR<sub>0</sub>~IR<sub>6</sub> 屏蔽掉, 屏蔽字为 01111111B=7FH, 写入奇地址。

⑤ 初始化程序段为:

```

CLI
MOV  AL,  13H      ; ICW1
OUT  8CH, AL
MOV  AL,  77H      ; ICW2
OUT  8DH, AL
MOV  AL,  01H      ; ICW4
OUT  8DH, AL
MOV  AL,  7FH      ; OCW1
OUT  8DH, AL
STI

```

(2) 中断类型码 77H 的中断向量设置程序。

假设相应的中断服务程序名为 INTP, 该符号地址包含段值属性和段内偏移量属性, 将二者分别存入中断向量地址: 中断类型码 77H 的中断向量地址为 77H×4=1DCH, 即占用 1DCH~1DFH 4 个单元, 其中 1DEH~1DFH 存放 INTP 的段地址, 1DCH~1DDH 存放 INTP 的段内偏移量。

我们用串指令完成中断向量的设置, 程序如下:

```

CLI
MOV  AX,  0
MOV  ES,  AX          ; 中断向量表段地址
MOV  DI,  1DCH        ; 中断向量表偏移地址
MOV  AX,  OFFSET INTP ; 中断服务程序偏移地址
CLD
STOSW
MOV  AX,  SEG  INTP    ; 中断服务程序段地址
STOSW
STI

```



## 习题

1. 什么叫中断？一般微机中包含有哪几类中断？它们各有什么特点？
2. 什么叫中断源？中断嵌套的含义是什么？
3. 通常 CPU 响应外部中断的条件有哪些？
4. 试简述 CPU 响应中断后，中断处理的过程。用流程图表示。
5. 什么情况下需要有中断判优机构？程序查询式和中断向量式两种中断源识别与判优方案各有什么特点？
6. 什么叫中断向量和中断向量号？试说明 8088/8086 可屏蔽硬中断是怎样获得中断向量，从而进入中断程序的。
7. 试说明 8259A 芯片的主要功能。
8. 8259A 芯片是一种什么类型的芯片？试说明 8259A 的体系结构。

# 第 8 章

## 可编程定时/计数器 8253

### 教学目的和要求

本章主要介绍定时/计数器 8253 的结构、各种工作方式、编程应用举例。重点掌握 8253 的结构，学会对定时/计数器编程和微机接口的定时控制。

在控制系统中，常常要求有一些实时时钟，以实现定时或延时控制，如定时中断、定时检测、定时扫描等；也往往要求有计数器，能对外部事件计数。

可编程定时器电路的定时值及其范围，可以很容易地由软件来确定和改变。所以，功能较强，使用灵活。本章中就介绍这种定时器电路。

Intel 系列的定时/计数器电路为可编程序间隔定时器 PIT（Programmable Interval Timer），型号为 8253，改进型为 8254。

## 8.1 概 述

8253 是 Intel 8080/8085 系列的可编程序间隔定时器，是一片采用 HN MOS 工艺，使用单一+5V 电源，24 引脚的双列直插式大规模集成电路。

8253 具有以下基本功能：

- ① 有 3 个独立的 16 位计数器；
- ② 每个计数器可按照二进制或十进制计数；
- ③ 每个计数器计数频率最高为 2.6MHz；
- ④ 每个计数器可选择 6 种不同的工作方式；
- ⑤ 所有的输入/输出与 TTL 兼容。

### 8.1.1 8253 的内部结构

8253 的内部结构如图 8-1 所示。它由 4 部分组成：数据总线缓冲器，读/写逻辑，控制字寄存器和 3 个定时/计数器通道。

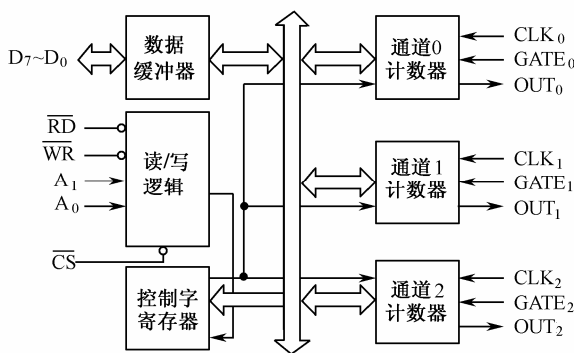


图 8-1 8253 的内部结构

#### 1. 数据总线缓冲器

这是一个三态、双向的 8 位缓冲器，用做系统数据总线和 8253 的接口，它根据 CPU 的输入或输出指令实现数据的输出或输入。这个数据总线缓冲器具有以下三个基本功能：

- ① 可以用程序设置 8253 的工作方式；
- ② 写入计数器计数初值；
- ③ CPU 从某一个通道读取的计数值。

## 2. 读/写控制逻辑

读/写逻辑接收系统控制总线的输入信号，然后依次产生整个器件操作所需要的控制信号，通过片选信号来控制读/写逻辑的工作。若没有被选中，则读/写逻辑操作功能不会发生变化。这时，8253 不再与 CPU 交换信息，数据总线缓冲器呈三态，芯片从系统数据总线上脱开。

## 3. 控制字寄存器

当  $A_1A_0$  的值为 11 时，选中控制字寄存器。

控制字寄存器在被选中的情况下可以接受 CPU 发来的控制信息，并将它存放在其中的寄存器中。用保存在控制字寄存器中的信息来控制每一个通道的工作方式，如选择二进制计数还是选择十进制计数，以及选择控制哪个通道等。控制字寄存器只能被写入，不能由 CPU 读出。

## 4. 计数器#0，计数器#1，计数器#2

这是三个计数/定时器通道，每一个都是由一个 16 位的可预置值的减法计数器构成的。这三个通道的操作是完全独立的。

每个通道都是对输入脉冲 CLK 按二进制或二—十进制方式，从预置值开始减 1 计数。当预置值减到 0 时，从 OUT 输出端输出一个信号。

### 8.1.2 8253 的引脚功能

8253 芯片的外部引脚如图 8-2 所示。

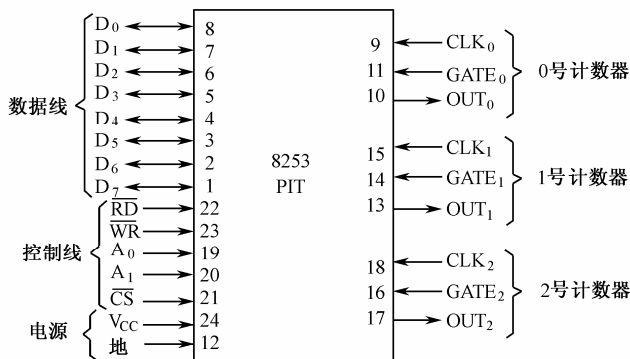


图 8-2 8253 引脚图

① 数据线  $D_7 \sim D_0$ ：与 CPU 的数据总线相连接，是双向三态的数据线，用于与 CPU 交换信息。

② 读信号  $\overline{RD}$ ： $\overline{RD}$  是一个低电平有效的由 CPU 发来的控制信号， $\overline{RD}$  信号通知 8253，CPU 要读取 8253 中的某个通道的计数值或状态字。也就是说由  $\overline{RD}$  来读取 8253 的某个计数器的相关内容。

③ 写信号  $\overline{WR}$ ： $\overline{WR}$  是一个低电平有效的由 CPU 发来的控制信号，CPU 通过此信号向 8253 发送控制字和计数值。

④ 片选信号  $\overline{CS}$ ： $\overline{CS}$  是一个低电平有效的信号，CPU 用此信号来选择 8253。在此芯

片不被选中的情况下，读信号和写信号没有意义，也不起作用。

⑤ 地址线  $A_1$ 、 $A_0$ ：这两条地址线一般接到系统地址总线的  $A_1$  和  $A_0$  上，它们的功能是编码选择 3 个通道和 1 个控制寄存器。其端口编码为：

$A_1$	$A_0$	端口
0	0	通道 0
0	1	通道 1
1	0	通道 2
1	1	控制寄存器

⑥  $CLK_2 \sim CLK_0$ ：3 个通道的外部时钟脉冲输入线，是 8253 计数器或定时器的计数脉冲，最大输入频率为 2MHz。

这个引脚输入的脉冲可以是系统时钟，也可以是由系统时钟分频或其他脉冲源提供的；可以是连续的，周期性的，均匀的，也可以是断续的，周期不定的，不均匀的。在一定程度上，这个输入时钟信号决定了通道是工作在计数方式还是定时方式。

⑦  $GATE_2 \sim GATE_0$ ：3 个通道的门控信号输入线，高电平有效，是用于启动或禁止通道工作的外部信号。通常，当  $GATE$  引脚为低电平时，禁止通道工作；当  $GATE$  引脚为高电平时，允许通道工作。

⑧  $OUT_2 \sim OUT_0$ ：3 个通道的计数为 0/定时时间到脉冲输出线。无论 8253 工作在何种方式，当减 1 计数器减 1 到 0 时，在  $OUT$  引脚上必定有输出，输出波形取决于 8253 的工作方式。

8253 内部端口的选择是由引线  $A_1$  和  $A_0$  决定的，他们通常接至地址总线的  $A_1$  和  $A_0$ 。各个通道的读/写操作的选择如表 8-1 所示。

表 8-1 8253 的端口选择

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	$A_1$	$A_0$	寄存器选择和操作
0	1	0	0	0	写入 0 号计数器
0	1	0	0	1	写入 1 号计数器
0	1	0	1	0	写入 2 号计数器
0	1	0	1	1	写入控制寄存器
0	0	1	0	0	读 0 号计数器
0	0	1	0	1	读 1 号计数器
0	0	1	1	0	读 2 号计数器
0	0	1	1	1	无操作（三态）
1	×	×	×	×	禁止（三态）
0	1	1	×	×	无操作（三态）

### 8.1.3 8253 的控制字

在 8253 的初始化编程中，由 CPU 向 8253 的控制字寄存器写入一个控制字，它规定了 8253 的工作方式。其格式如图 8-3 示。

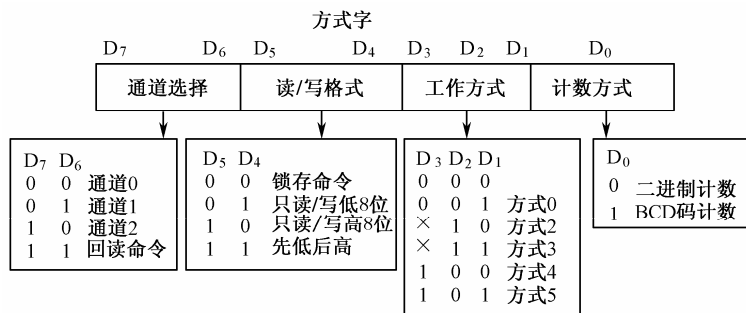


图 8-3 8253 的控制字格式

### 1. 计数器选择 (D<sub>7</sub>D<sub>6</sub>)

控制字的最高两位决定这个控制字是哪一个通道的控制字。由于三个通道的工作是完全独立的，所以需要有三个控制字寄存器分别规定相应通道的工作方式。但它们的地址是同一个，即 A<sub>1</sub>A<sub>0</sub>=11 为控制字寄存器的地址。所以，需要由这两位来决定是哪一个通道的控制字。因此，对三个通道的编程需要向同一个地址（控制字寄存器地址）写入三个控制字，它们的 D<sub>7</sub>D<sub>6</sub> 位分别指定不同的通道。在控制字中的通道选择与通道计数器的地址是两回事，不能混淆。计数通道的地址用做 CPU 向计数器写初值，或从计数器读取计数的当前值。

### 2. 数据读/写格式 (D<sub>5</sub>D<sub>4</sub>)

CPU 向计数通道写入初值和读取它们的当前状态时，有几种不同的格式。例如，写数据时，是写入 8 位数据还是 16 位数据：若是 8 位计数，可以令 D<sub>5</sub>D<sub>4</sub>=01 只写低 8 位，则高 8 位自动置 0；若是 16 位计数，而低 8 位为 0，则可令 D<sub>5</sub>D<sub>4</sub>=10，只写入高 8 位，低 8 位就自动为 0；在令 D<sub>5</sub>D<sub>4</sub>=11 时，就先写入低 8 位，后写入高 8 位。

在读取计数值时，可令 D<sub>5</sub>D<sub>4</sub>=00，则把写控制字时的计数值锁存，以后再读取。

### 3. 工作方式 (D<sub>3</sub>D<sub>2</sub>D<sub>1</sub>)

8253 的每个通道可以有 6 种不同的工作方式，由这三位决定。每一种方式的特点，随后介绍。

### 4. 计数方式 (D<sub>0</sub>)

8253 的每个通道有两种计数制：二进制和二十进制（BCD 码），由 D<sub>0</sub> 位决定。在二进制计数时，写入的初值的范围为 0000H~FFFFH，其中 0000H 时为最大值，代表 65536。在二十进制时，写入的初值的范围为 0000~9999，其中，0000 是最大值，代表 10000。

## 8.1.4 8253 的工作方式

### 1. 工作方式 0——计完最后一个数时中断

在这种方式下，当控制字 CW（Control Word）写入控制字寄存器时，使 OUT 端的输出变低，即使没有给计数器赋初值，也不计数。要开始计数，GATE 信号必须为高电平。在写入计数初值后，通道开始计数，在计数过程中 OUT 端的输出一直维持为低电平，直到计数到“0”时，OUT 端的输出变高。其过程如图 8-4 所示。其中，LSB=4 表示只写低 8

位，计数值为 4。最底下一行是计数器中的数值。

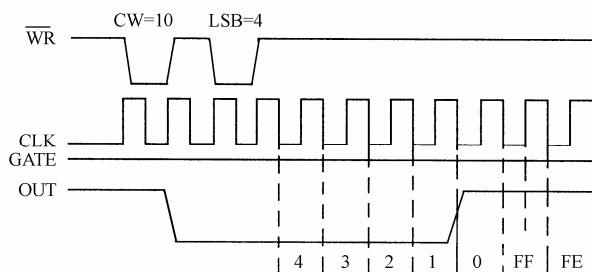


图 8-4 8253 工作方式 0 的波形图

若 8253 的地址为 04H~07H，要使计数器 1 工作在方式 0，仅用 8 位二进制计数，计数值为 128。初始化程序为：

```
MOV    AL, 50H    ; 设置控制字
OUT     07H, AL    ; 输出至控制字寄存器
MOV     AL, 80H    ; 计数值
OUT     05H, AL    ; 输出至计数通道 1
```

## 2. 工作方式 1——可程序的单拍脉冲

在这种方式下，当 CPU 写控制字之后（ $\overline{\text{WR}}$  的上升沿），输出将保持为高电平（若原为低电平，则由低变高）。当 CPU 写完计数值后，计数器并不开始计数，直到外部门控脉冲 GATE 启动之后的下一个输入 CLK 脉冲的下降沿才开始计数，输出信号 OUT 变为低电平。在整个计数过程中，OUT 都维持为低电平，直到计数到 0，输出变为高电平。因此，输出为一个单拍脉冲。若外部门控脉冲 GATE 再次触发启动，则可以再产生一个单拍脉冲，如图 8-5 所示。

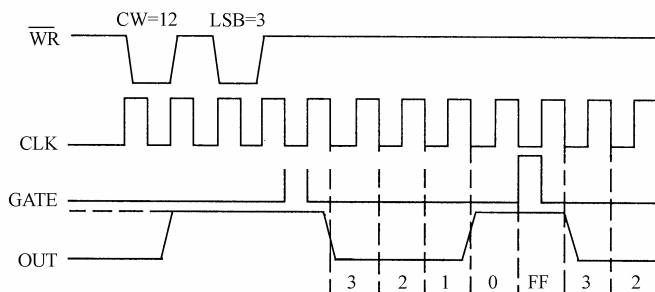


图 8-5 8253 工作方式 1 的波形图

若要使计数器 0 工作在方式 1，按 BCD 计数，计数值为 3000，则初始化程序为：

```
MOV     AL, 23    ; 设置方式控制字
OUT      07H, AL   ; 输出至控制字寄存器
MOV     AL, 30H    ; 设计输出值
OUT      04H, AL   ; 输出计数初值高 8 位
```

### 3. 工作方式 2——速率发生器

在这种方式下，当 CPU 输出控制字后，输出将为高电平。在写入计数值后，计数器将立即自动对输入时钟 CLK 计数。在计数过程中输出始终保持为高电平，直至计数值减到 1 时，输出将变为低电平。经过一个 CLK 周期，输出恢复为高电平，且计数器开始重新计数，如图 8-6 所示。

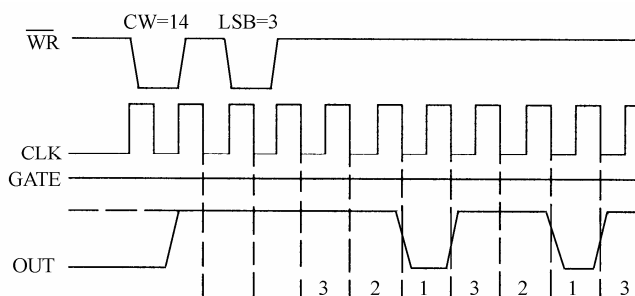


图 8-6 8253 工作方式 2 的波形图

工作方式 2 的一个突出特点是能够连续工作。如果计数值为  $N$ ，则每输入  $N$  个 CLK 脉冲，会输出一个脉冲。因此，这种方式可以作为一个脉冲速率发生器或用于产生实时时钟中断。

若要使计数器 2 工作于方式 2，按二进制计数，计数值为 02F0H，则初始化程序为：

```
MOV     AL, 0B4H
OUT     07H, AL      ; 写入控制字
MOV     AL, 0F0H
OUT     06H, AL      ; 写计数值的低 8 位
MOV     AL, 02H
OUT     06H, AL      ; 写计数值的高 8 位
```

### 4. 工作方式 3——方波速率发生器

工作方式 3 和工作方式 2 的输出都是周期性的，它们的主要区别是，工作方式 3 在计数过程中输出有一半时间为高电平，另一半时间为低电平。所以，若计数值为  $N$ ，则方式 3 的输出是周期为  $N$  个 CLK 脉冲的方波。

在这种方式下，当 CPU 设置控制字后，输出将为高电平。在写完计数值后就自动开始计数，输出保持为高电平；当计数到一半的计数值时，输出变为低电平，直至计数到 0，输出又变为高电平，重新开始计数，如图 8-7 所示。若  $N$  为偶数时，OUT 信号的输出有  $N/2$  个 CLK 脉冲为高电平，有  $N/2$  个 CLK 脉冲为低电平。若  $N$  为奇数时，OUT 输出有  $(N+1)/2$  个 CLK 脉冲为高电平，有  $(N-1)/2$  个 CLK 脉冲为低电平。

### 5. 工作方式 4——软件触发选通

在这种方式下，当写入控制字后，输出为高电平（原为高则保持为高，原为低则变为高）。当写入计数值后立即开始计数（相当于软件启动），当计数到 0 后，输出变为低电平，经过一个输入时钟周期，输出又变为高电平，计数器停止计数。故这种方式的计数也是一次性的，只有在输入新的计数值后，才能开始新的计数，如图 8-8 所示。



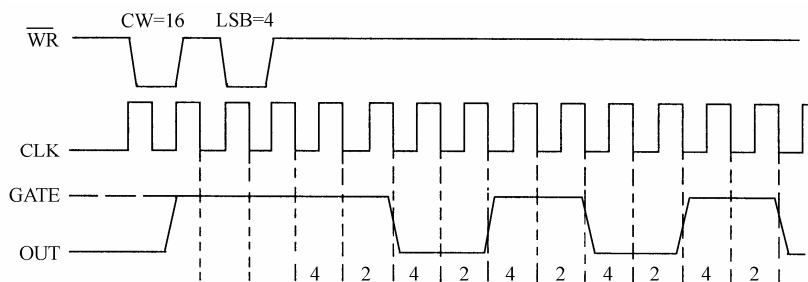


图 8-7 8253 工作方式 3 的波形图

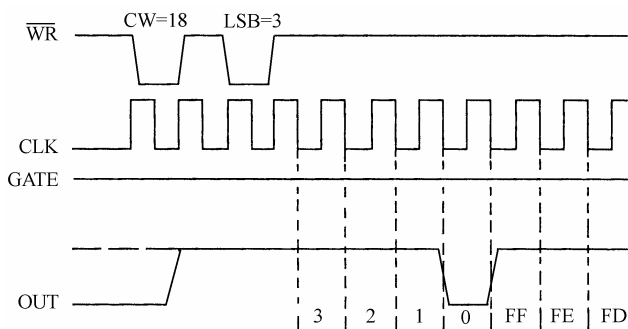


图 8-8 8253 工作方式 4 的波形图

## 6. 工作方式 5——硬件触发选通

在这种方式下，设置了控制字后，输出为高电平。在设置了计数值后，计数器并不立即开始计数，而是由门控脉冲的上升沿触发启动。当计数到 0 时，输出变为低电平；经过一个 CLK 脉冲，输出恢复为高电平，停止计数，要等到下次门控脉冲的触发才能再计数，如图 8-9 所示。

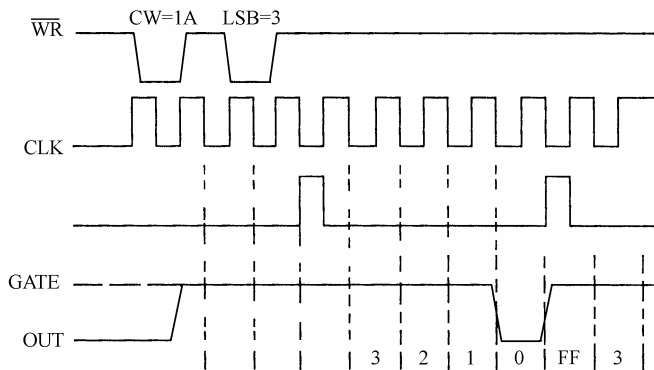


图 8-9 8253 工作方式 5 的波形图

## 8.2 8253 的编程

要使用 8253 必须首先进行初始化编程。初始化编程的内容为：先写入每一个通道的控制字，然后再写入通道的计数初值。如前所述，在有些方式下，写入计数初值后此计数通道就开始工作了；而有的方式则需要外部门控信号的触发启动。

在初始化编程时，某一通道的控制字和计数初值，是通过两个不同的端口地址写入的。任一通道的控制字都将写入控制字寄存器（地址总线低两位  $A_1A_0=11$ ），由控制字中的  $D_7D_6$  来确定是哪一个通道的控制字；而计数初值是由各个通道的端口地址写入的。

初始化编程的步骤为：

(1) 写入通道控制字，规定通道的工作方式。

(2) 写入计数初值。

① 若规定只写低 8 位，则写入的为计数值的低 8 位，高 8 位自动置 0。

② 若规定只写高 8 位，则写入的为计数值的高 8 位，低 8 位自动置 0。

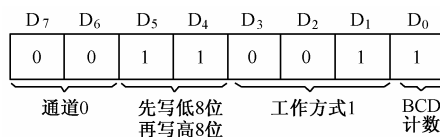
③ 若是 16 位的计数值，则分两次写入，先写入低 8 位，再写入高 8 位。

若要用通道 0，工作在方式 1，按二—十进制计数，计数值为 5080H。则初始化编程的步骤为：

① 确定通道控制字；

② 计数值的低 8 位为 80H；

③ 计数值的高 8 位为 50H。



若端口地址为 F8H~FBH，则初始化程序为：

```
MOV    AL, 33H
OUT    0FBH, AL
MOV    AL, 80H
OUT    0F8H, AL
MOV    AL, 50H
OUT    0F8H, AL
```

8253 任一通道的计数值，CPU 可用输入指令读取。CPU 读取的是执行输入指令瞬间计数器的现行值。但 8253 的计数器是 16 位的，所以要分两次读至 CPU。因此，若不设法锁存的话，则在输入过程中，计数值可能已经变化了。要锁存有两种办法：

① 利用 GATE 信号使计数过程暂停。

② 向 8253 输送一个通道控制字，令 8253 通道中的锁存器锁存。8253 的每一个通道都有一个输出锁存器（16 位），平时，它的值随通道计数器的值变化，当向通道写入锁存的控制字时，它把计数器的现行值锁存（计数器可继续计数），于是 CPU 读取的就是锁存器中的值。当对计数器重新编程，或 CPU 读取了计数值后，自动解除锁存状态，它的值又随计数器变化。

若要读取通道的 16 位计数值，其程序为：

MOV	AL, 40H	; 计数器 1 的锁存命令
OUT	0FBH, AL	; 写入至控制字寄存器
IN	AL, 0F9H	; 读低 8 位
MOV	CL, AL	; 存于 CL 中
IN	AL, 0F9H	; 读高 8 位
MOV	CH, AL	; 存于 CH 中

## 习题

1. 8253 有几个定时/计数器通道？
2. 8253 的 CLK、GATE 信号有什么作用？
3. 简述 8253 的主要功能和内部结构。
4. 在某一 8088 CPU 系统中，8253 的控制端口地址为 43H，通道 0、通道 1、通道 2 的端口地址分别为 40H、41H、42H。通道 0 工作在方式 2，计数器初值为 1000；通道 1 工作在方式 3，计数器初值为 65530；通道 2 工作在方式 1，计数器初值为 10000。试编写程序实现 8253 的初始化。
5. 在某一 8088 CPU 系统中，8253 的控制端口地址为 73H，通道 0、通道 1、通道 2 的端口地址分别为 70H、71H、72H，在通道 0 的 CLK 端输入 1MHz 的方波。试编写程序实现在 8253 通道 0 的 OUT 端输出 1kHz 的方波。
6. 在某一 8088 CPU 系统中，8253 的控制端口地址为 83H，通道 0、通道 1、通道 2 的端口地址分别为 80H、81H、82H，在通道 0 的 CLK 端输入 1.4MHz 频率的方波，在 OUT 端连接一扬声器。试编写程序实现让扬声器发出 500Hz 的声音。

# 第9章

## 接 □ 电 路

### 📖 教学目的和要求

本章主要介绍在微机应用系统中常用到的一些典型的可编程接口芯片，数/模、模/数转换芯片结构、工作原理及其接口硬件设计和软件设计。要求掌握可编程接口 8255A、DAC0832、ADC0809 结构特性、应用方法。

## 9.1 可编程并行接口 8255A

并行接口一般具有两个或两个以上的 8 位 I/O 接口。各个 I/O 接口的工作方式可由程序分别确定或改变，使用灵活，便于和各种外部设备连接。因此，又称可编程的外部接口（PPI）。目前各主要微处理器厂商都有自己的 PPI 产品，但它们的功能基本类似。下面介绍的是 Intel 的 PPI 产品——8255A 可编程外部设备接口。该芯片可以和 8086、8088 等微处理器直接连接，也适用于 MCS-48、MCS-51 系列单片机。

### 9.1.1 8255A 的结构

8255A 是 Intel 公司生产的可编程输入/输出接口芯片，它具有 3 个 8 位的并行 I/O 口，分别为 PA 口、PB 口和 PC 口（简称 A 口、B 口、C 口），其中 PC 口又分为高 4 位口（PC<sub>7</sub>~PC<sub>4</sub>）和低 4 位口（PC<sub>3</sub>~PC<sub>0</sub>），它们都可以通过软件编程来改变 I/O 口的工作方式。

8255A 的引脚图如图 9-1 所示。8255A 的结构框图如图 9-2 所示。

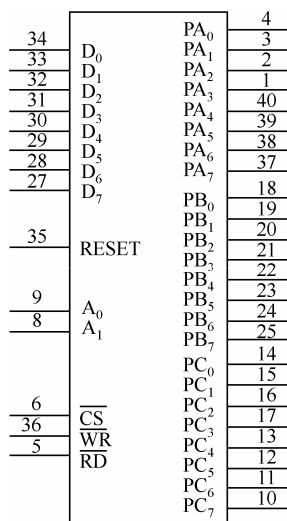


图 9-1 8255A 的引脚图

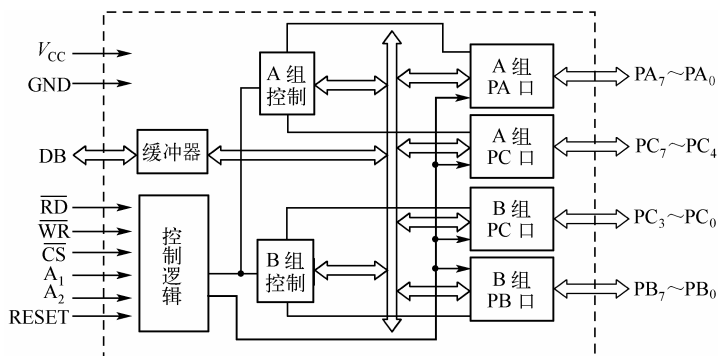


图 9-2 8255A 的结构图

它由以下几个部分组成：

#### 1. 数据端口

8255A 的 3 个并行口都可以选择作为输入/输出的工作模式，但在功能和结构上有些差异。

PA 口：一个 8 位数据输出锁存器和缓冲器；一个 8 位数据输入锁存器。

PB 口：一个 8 位数据输出锁存器和缓冲器；一个 8 位数据输入缓冲器。

PC 口：一个 8 位输出锁存器；一个 8 位数据输入缓冲器。

通常 PA 口、PB 口作为输入/输出口，PC 口可作为输入/输出口，也可在软件的控制

下,分为两个4位的端口,作为PA、PB端口选通方式操作时的状态控制信号。

## 2. A组和B组控制电路

这是两组根据CPU写入的“命令字”控制8255A工作方式的控制电路。A组控制PA口和PC口的上半部(PC<sub>7</sub>~PC<sub>4</sub>);B组控制PB口和PC口的下半部(PC<sub>3</sub>~PC<sub>0</sub>)。

## 3. 双向三态数据缓冲器

这是8255A和CPU数据总线的接口,CPU和8255A之间的命令、数据和状态的传递都通过双向三态总线缓冲器传送,D<sub>7</sub>~D<sub>0</sub>接CPU的数据总线。

## 4. 读写和控制逻辑

A<sub>0</sub>、A<sub>1</sub>、 $\overline{\text{CS}}$ 为8255A的端口选择信号和片选信号, $\overline{\text{RD}}$ 、 $\overline{\text{WR}}$ 为8255A的读写控制信号,这些信号线分别和MCS-51的地址线和读写信号线相连接,实现CPU对8255A口的选择和数据传送。

CPU对8255A的A口、B口、C口和控制口的寻址,如表9-1所示。

表 9-1 8255A 端口选择表

操 作	$\overline{\text{CS}}$	A <sub>1</sub>	A <sub>0</sub>	$\overline{\text{RD}}$	$\overline{\text{WR}}$	功 能
输入	0	0	0	0	1	A口→数据总线(读端口A)
输入	0	0	1	0	1	B口→数据总线(读端口B)
输入	0	1	0	0	1	C口→数据总线(读端口C)
输入	0	1	1	0	1	状态寄存器→数据总线
输出	0	0	0	1	0	数据总线→A口(写端口A)
输出	0	0	1	1	0	数据总线→B口(写端口B)
输出	0	1	0	1	0	数据总线→C口(写端口C)
输出	0	1	1	1	0	数据总线→控制寄存器
禁止	1	×	×	×	×	数据总线为高阻态

## 5. 复位控制

引脚RESET为复位信号输入脚,高电平有效。复位有效时,它把控制寄存器清零和置所有端口(A、B、C)为输入方式。

### 9.1.2 8255A的工作方式

8255A有3种基本工作方式:方式0——基本输入/输出;方式1——选通输入/输出;方式2——双向传送(仅PA口)。工作方式的选择由CPU输出的控制字决定。

#### 1. “方式”选择控制字

8255A的工作方式,可由CPU送出一个控制字到8255A的控制字寄存器来选择。这个控制字的格式如图9-3所示,可以分别选择端口A和端口B的工作方式,端口C分成两部分,上半部分随端口A,下半部随端口B。端口A可工作于以上3种工作方式,而端口B只能工作于方式0和方式1。最高位D<sub>7</sub>是该控制字的标志位,其状态固定为1,用于表明本字节是方式控制字。

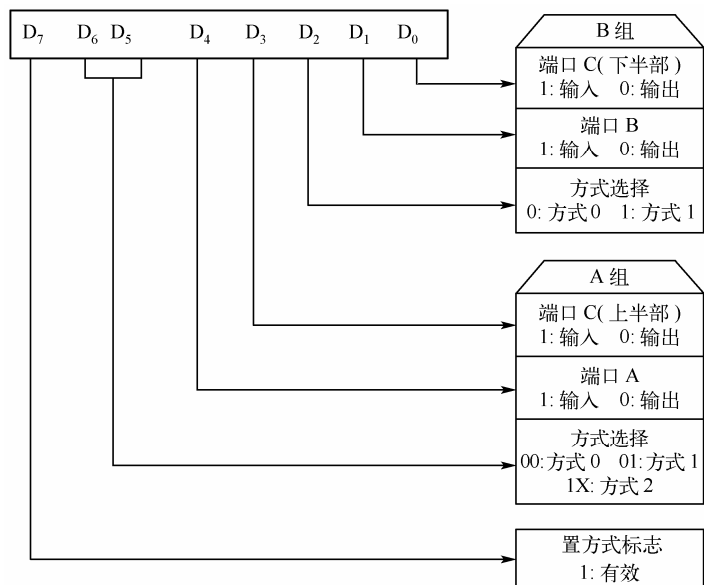


图 9-3 8255A 的控制字

【例 9-1】 若对 8255A 做如下设置：A 口方式 0 输入，B 口方式 1 输出，C 口高位部分为输出、低位部分为输入。设控制寄存器地址为 0FFF<sub>BH</sub>。

按各口的设置要求，工作方式控制字为 10010101<sub>B</sub>，即 95<sub>H</sub>。则初始化程序段为

```
MOV DX, 0FFFBH
MOV AL, 95H
OUT DX, AL
```

## 2. C口按位置位/复位功能

端口 C 的 8 位中的任意一位，可用一个写入 8255A 的控制口的置位/复位控制字来置位或复位。这个功能主要用于控制。控制字的格式如图 9-4 所示。D7 是该控制字的标志，其状态固定为 0。

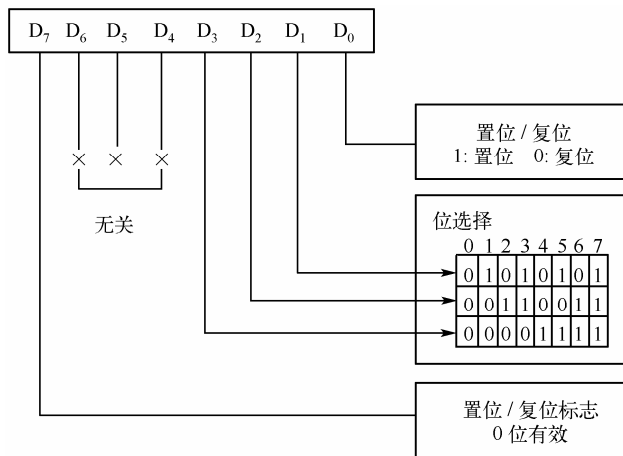


图 9-4 端口 C 按位置位/复位控制字

【例 9-2】 如果想把 8255A 的 C 口的 PC<sub>1</sub> 置 1, PC<sub>7</sub> 复位, 该如何对 8255A 编程呢。将 03H 写入控制口, 置 “1” PC<sub>1</sub>; 0EH 写入控制口, 清零 PC<sub>7</sub>。设控制寄存器地址为 0FFFBH。程序如下。

```
MOV    DX, 0FFFBH
MOV    AL, 03H
OUT    DX, AL
MOV    AL, 0EH
OUT    DX, AL
```

### 3. 方式 0 的功能

方式 0 是一种基本的输入/输出方式。在这种工作方式下, 3 个端口的每一个都可由程序选定作为输入或输出, 这种方式适用于无条件地传送数据的设备。例如, 读一组开关的状态, 控制一组指示灯的亮与灭, 并不需要联络信号, CPU 可随时读入开关的状态, 随时可把一组数据送到指示灯显示。

方式 0 的基本功能为:

- ① 两个 8 位端口 (A 和 B) 和两个 4 位端口 (C)。
- ② 任意一个端口都可以作为输入或输出。
- ③ 输出是锁存的。
- ④ 输入是不锁存的。
- ⑤ 在方式 0 时, 各个端口的输入、输出可有 16 种不同的组合。

在这种工作方式下, 由于是无条件的传送, 所以不需要状态端口, 3 个端口都可作为数据端口。

【例 9-3】 用 8255A 产生波形。8255A 在方式 0 下工作, 令其在 PB<sub>1</sub>、PB<sub>2</sub> 引脚产生如图 9-5 所示波形, 试编写相应程序。电路连接设定 8255A 各端口地址分别为 90H、91H、92H 和 93H, 波形延时时间可调用延时 1 毫秒 (D1ms) 子程序实现。

根据要求可确定端口 B 应工作在方式 0 下输出, 其余端口无具体要求, 也都定为方式 0 输出, 那么方式选择控制字为 80H。程序如下:

```
MOV    AL, 80H    ; 8255A 的初始化
OUT    93H, AL
START: MOV    AL, 02H
OUT    91H, AL
CALL   D1ms
MOV    AL, 06H
OUT    91H, AL
CALL   D1ms
MOV    AL, 00H
OUT    91H, AL
CALL   D1ms
```

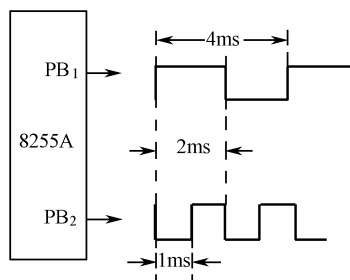


图 9-5 8255A 产生波形接口电路



```
MOV AL, 04H
OUT 91H, AL
CALL D1ms
JMP START
```

4. 方式 1 的功能

这是一种选通的 I/O 方式。在这种工作方式时，端口 A 或端口 B 作为数据的输入/输出，但同时规定端口 C 的某些位作为数据传送的联络信号，具体定义见表 9-2。方式 1 适用于查询或中断方式的数据输入/输出。

表 9-2 C 口联络信号定义

C 口位线	方 式 1		方 式 2	
	输 入	输 出	输 入	输 出
PC <sub>7</sub>		$\overline{\text{OBFA}}$		$\overline{\text{OBFA}}$
PC <sub>6</sub>		$\overline{\text{ACKA}}$		$\overline{\text{ACKA}}$
PC <sub>5</sub>	IBFA		IBFA	
PC <sub>4</sub>	$\overline{\text{STBA}}$		$\overline{\text{STBA}}$	
PC <sub>3</sub>	INTRA	INTRA	INTRA	INTRA
PC <sub>2</sub>	$\overline{\text{STBB}}$	$\overline{\text{ACKB}}$		
PC <sub>1</sub>	IBFB	$\overline{\text{OBFB}}$		
PC <sub>0</sub>	INTRB	INTRB		

(1) 方式 1 的基本功能

① 用做一个或两个选通端口。

② 每一个端口包含有：8 位数据端口；三条控制线（是固定指定的，不能用程序改变）；提供中断逻辑。

③ 任何一个端口都可以作为输入或输出。

④ 若只有一个端口工作于方式 1，余下的 13 位，可以工作在方式 0（由控制字决定）。

⑤ 若两个端口都工作于方式 1，端口 C 还留下两位，这两位可以由程序指定作为输入或输出，也具有置位/复位功能。

(2) 方式 1 输入

当任意一个端口工作于方式 1 输入时，其逻辑组态如图 9-6 所示。其各个控制信号的意义为：

①  $\overline{\text{STB}}$  (Strobe)：选通脉冲（输入），低电平有效。这是由外设供给的输入控制信号，当其有效时，把从外设来的数据送入锁存器。

② IBF (Input Buffer Full)：输入缓冲器满信号（输出），高电平有效。这是一个 8255A 输出的状态信号，当其有效时，表示数据已输入至输入锁存器，它由 STB 信号置位（高电平），而读信号的上升沿使其复位。

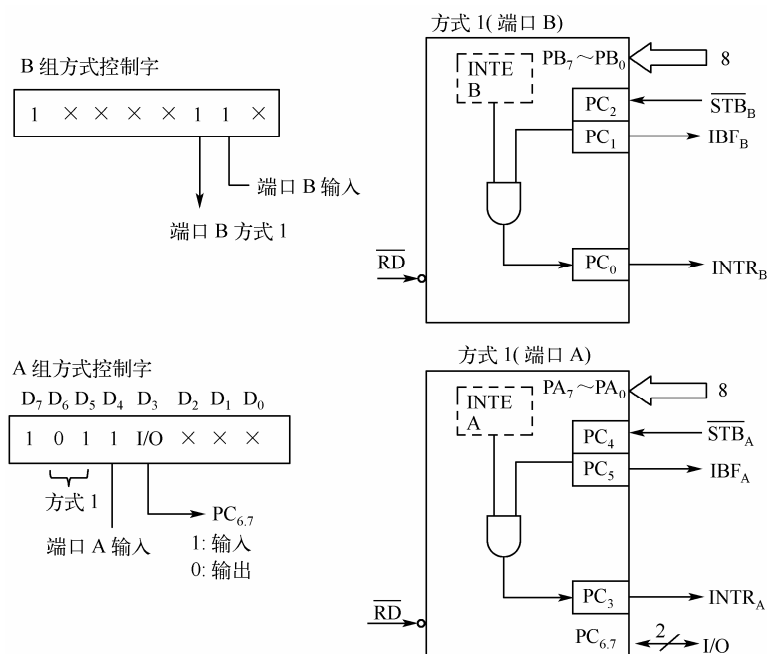


图 9-6 方式 1 输入组态

③ **INTR(Interrupt Request)**: 中断请求信号(输出), 高电平有效。这是 8255A 的一个输出信号, 可用做向 CPU 的中断请求信号, 以要求 CPU 服务。当  $\overline{STB}$ 、 $\overline{IBF}$  为高电平, 同时  $\overline{INTE}$  (中断允许) 被置为 1 时,  $\overline{INTR}$  变为高电平, 而由  $\overline{RD}$  信号的下降沿清除。

④  $\overline{INTEA}$  和  $\overline{INTEB}$  为中断使能信号。

### (3) 方式 1 输出

方式 1 输出时, 其逻辑组态如图 9-7 所示, 主要的控制信号如下:

①  $\overline{OBF}$  (**Output Buffer Full**): 输出缓冲器满信号, 低电平有效。这是 8255A 输出给外设的一个控制信号。当其有效时, 表示 CPU 已经把数据输出给指定的端口, 可以把数据输出。它由输出命令的  $\overline{WR}$  上升沿置成低电平, 由  $\overline{ACK}$  的有效信号使其恢复为高电平。

②  $\overline{ACK}$  (**Acknowledge**): 低电平有效。这是一个外设的响应信号, 指示 CPU 输出给 8255A 的数据已经由外设接收。

③ **INTR**: 中断请求信号, 高电平有效。当输出装置已经接收 CPU 输出的数据后, 用它来向 CPU 提出新的中断请求, 要求 CPU 继续输出数据。当  $\overline{ACK}$  为“1 (高电平)”,  $\overline{OBF}$  为“1 (高电平)”和  $\overline{INTE}$  为“1 (高电平)”时, 使其置位 (高电平), 而  $\overline{WR}$  信号的下降沿使其复位 (低电平)。

④  $\overline{INTEA}$ : 由  $PC_6$  的置位/复位控制。

⑤  $\overline{INTEB}$ : 由  $PC_2$  的置位/复位端控制。

### 5. 方式 2 的功能

这种工作方式, 使外设可在单一的 8 位总线上, 既能发送也能接收数据 (双向总线 I/O)。工作时可用程序查询方式, 也可工作于中断方式。

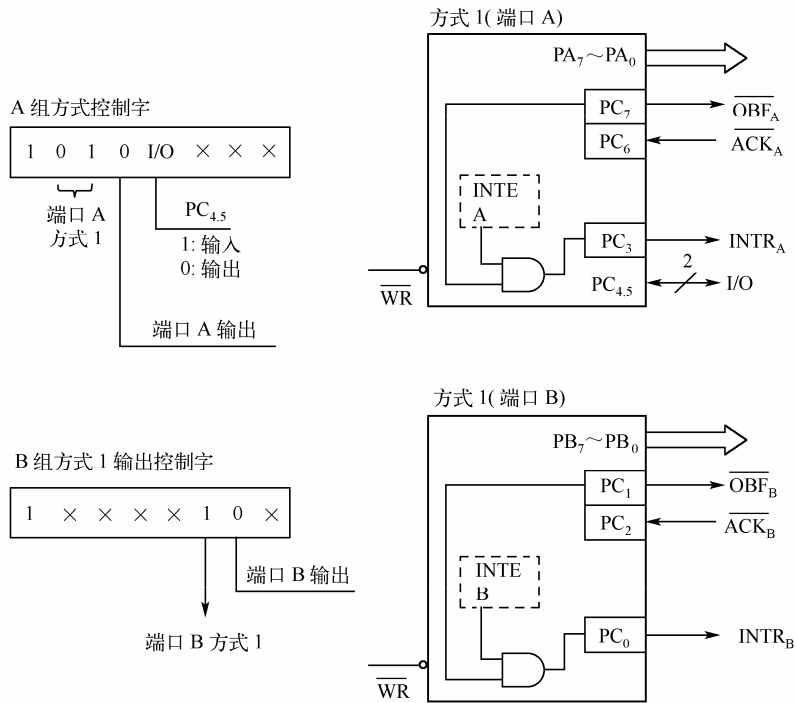


图 9-7 方式 1 输出组态

- 该工作方式的主要功能：
- ① 方式 2 只用于端口 A，端口 B 无此种工作方式。
  - ② 一个 8 位的双向总线端口（端口 A）和一个 5 位控制端口（端口 C）。
  - ③ 输入和输出是锁存的。
  - ④ 5 位控制端口用做端口 A 的控制和状态信息。
- 8255A 工作在方式 2 时，其逻辑组态如图 9-8 所示。各个信号的意义为：

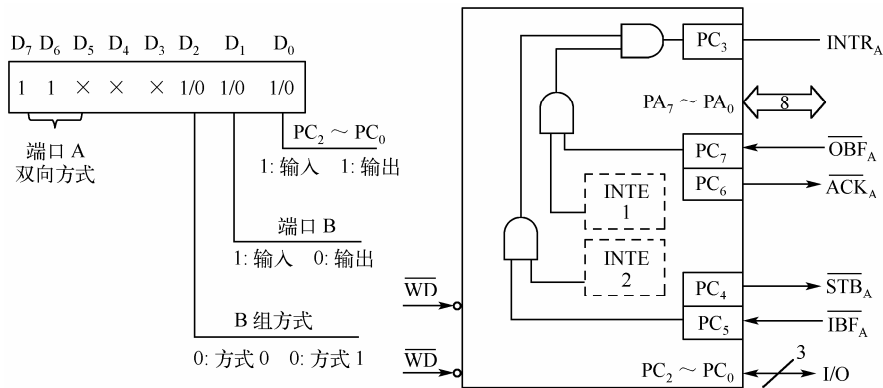


图 9-8 方式 2 输出组态

①  $\overline{\text{INTR}}$ : 中断请求, 高电平有效。在输入和输出方式时, 都可用做向 CPU 的中断请求信号。

②  $\overline{\text{OBF}}$ : 输出缓冲器满, 低电平有效。它是对外设的一种选通信号, 表示 CPU 已把数据输出至端口 A。

③  $\overline{\text{ACK}}$ : 响应信号, 低电平有效。它启动端口 A 的三态输出缓冲器, 送出数据; 否则, 输出缓冲器处在高阻状态。

④  $\overline{\text{INTE1}}$ : 与输出缓冲器相关的中断屏蔽触发器, 由 PC6 的置位/复位控制。

⑤  $\overline{\text{STB}}$ : 选通输入, 低电平有效。这是外设供给 8255A 的选通信号, 它把输入数据选通至 8255A 的输入锁存器。

⑥  $\overline{\text{IBF}}$ : 输入缓冲器满, 高电平有效。它是一个状态信息, 指示数据已进入输入锁存器。

⑦  $\overline{\text{INTE2}}$ : 与输入缓冲器相关的中断屏蔽触发器, 由 PG4 的置位/复位控制。

## 9.2 可编程多功能接口 8155

8155 芯片内具有 256 个字节的 RAM, 2 个 8 位、1 个 6 位的可编程 I/O 口和 1 个 14 位计数器。

### 9.2.1 8155 的结构及引脚

8155 的逻辑结构及引脚如图 9-9 所示。

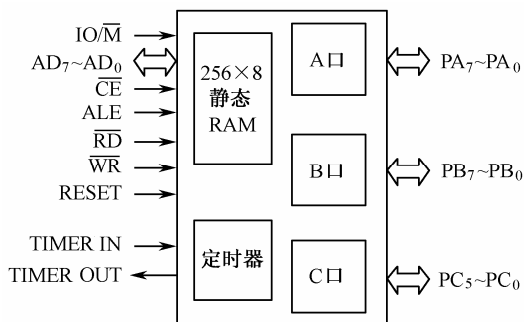


图 9-9 8155 的逻辑结构及引脚

$\text{AD}_7 \sim \text{AD}_0$  为地址数据总线, CPU 和 8155 之间的地址、数据、命令、状态信息都通过这个总线传送。

ALE 为地址锁存信号输入线。在 ALE 的下降沿将 CPU 输出的低 8 位地址信息锁存到 8155 内部寄存器。 $\text{IO}/\overline{\text{M}}$  为 RAM/I/O 口选择线。当  $\text{IO}/\overline{\text{M}}=0$  时, CPU 选择 8155 的 RAM 读/写,  $\text{AD}_7 \sim \text{AD}_0$  地址为 8155 中 RAM 单元地址; 当  $\text{IO}/\overline{\text{M}}=1$  时, CPU 选择 8155 的 I/O 口,  $\text{AD}_7 \sim \text{AD}_0$  地址为 I/O 口地址。8155 内部 I/O 口编址如表 9-1 所示。

$\overline{\text{CE}}$  为片选信号,  $\overline{\text{RD}}$ 、 $\overline{\text{WR}}$  为读、写控制输入线。

表 9-1 8155 内部 I/O 口编址

A7	A6	A5	A4	A3	A2	A1	A0	I/O 口
×	×	×	×	×	0	0	0	命令状态寄存器（命令/状态口）
×	×	×	×	×	0	0	1	PA 口
×	×	×	×	×	0	1	0	PB 口
×	×	×	×	×	0	1	1	PC 口
×	×	×	×	×	1	0	0	定时器低 8 位
×	×	×	×	×	1	0	1	定时器高 8 位

### 9.2.2 8155 的工作方式与基本操作

8155 可作为 I/O 口、片外 256 字节数据存储器及定时器使用。

#### 1. 256 字节数据存储器

将 8155 的  $\text{IO}/\overline{\text{M}}$  引脚置低电平，这时 8155 只能用做片外 256 字节数据存储器，与应用系统中其他数据存储器统一编址。

#### 2. 扩展 I/O 口

8155 用做扩展 I/O 口时， $\text{IO}/\overline{\text{M}}$  引脚必须置高电平，这时 PA、PB、PC 口的口地址的低 8 位分别为 01H、02H、03H（设地址无关位为 0）。

8155 的 I/O 口工作方式选择是通过 8155 内部命令寄存器（命令口）设定命令控制字来实现的。

##### （1）I/O 口的工作方式选择

8155 的 A 口、B 口可工作于基本 I/O 方式或选通方式，C 口可作为输入/输出口，也可以作为 A 口、B 口选通方式工作时的状态控制信号线。I/O 口工作方式选择完全依靠对 8155 命令寄存器（命令口）设定命令控制字的方式实现。命令寄存器只能写入不能读出。命令寄存器格式如图 9-10 所示。

8155 有 4 种工作方式：

方式 1：基本 I/O 工作方式，A 口、B 口均为基本输入/输出方式，C 口为输入方式。

方式 2：基本 I/O 工作方式，A 口、B 口均为基本输入/输出方式，C 口为输出方式。

方式 3：A 口定义为选通 I/O，B 口定义为基本 I/O，其中， $\text{PC}_0=\text{AINTR}$ ， $\text{PC}_1=\text{ABF}$ ， $\text{PC}_2=\text{ASTB}$ ， $\text{PC}_3\sim\text{PC}_5$  输出。

方式 4：A 口、B 口都定义为选通 I/O，其中， $\text{PC}_0=\text{AINTR}$ ， $\text{PC}_1=\text{ABF}$ ， $\text{PC}_2=\text{ASTB}$ ， $\text{PC}_3=\text{BINTR}$ ， $\text{PC}_4=\text{BBF}$ ， $\text{PC}_5=\text{BSTB}$ 。

INTR 为中断请求输出线，高电平有效。当 8155 的 A 口（或 B 口）缓冲器接收到设备输入的数据或设备从缓冲器中取走数据时，中断请求线 INTR 的电位升高（仅当命令寄存器相应中断允许位为 1 时），向 CPU 请求中断，CPU 对 8155 的相应 I/O 口进行一次读/写操作，INTR 变为低电平。

BF 为缓冲器状态标志输出线。缓冲器有数据时 BF 为高电平，否则为低电平。

STB 为设备选通信号输入线，低电平有效。

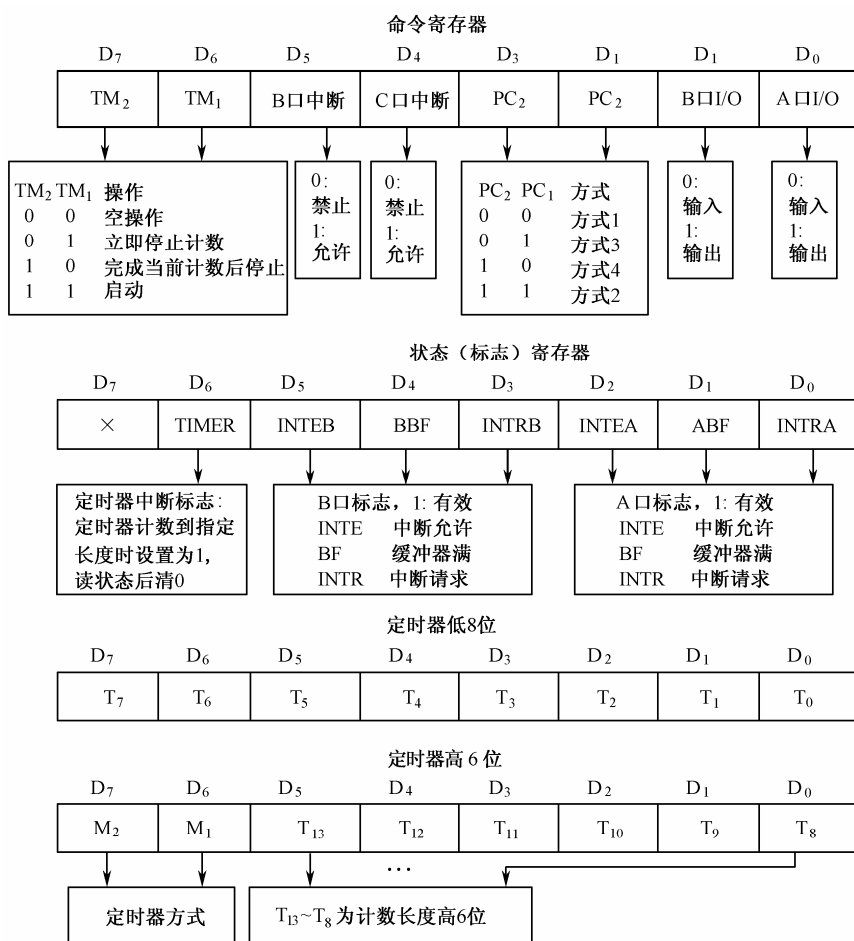


图 9-10 8155 控制字格式

### (2) I/O的状态查询





8155 有一个状态寄存器，用于锁存 I/O 口和定时器的当前状态，供 CPU 查询。状态寄存器和命令寄存器公用一个地址，只能读出不能写入。因此，可以认为 8155 的 00H 口是命令/状态寄存器，对其写入时作为命令寄存器，写入的是命令；而对其读出时，作为状态寄存器，读出的是当前 I/O 口和定时器的状态。

## 3. 定时器扩展

### (1) 定时器的方式选择

8155 片内有一个 14 位减法计数器，可对输入脉冲进行减法计数。其外部有两个定时器引脚端 TIN、TOUT。TIN 为定时器时钟输入，由外部输入时钟脉冲；TOUT 为定时器输出，输出各种信号波形。定时器的 14 位计数器由 04H 端口（低 8 位）和 05H 端口的 D<sub>5</sub>~D<sub>0</sub>（高 6 位）组成。定时器输出有四种波形，可由定时器方式编程选择。定时器方式及相应的输出波形如表 9-2 所示。

表 9-2 8155 定时器方式及输出波形

M <sub>2</sub>	M <sub>1</sub>	方 式	定时器输出波形
0	0	单方波	
0	1	连续方波	
1	0	单脉冲	
1	1	连续脉冲	

(2) 定时器的编程

对定时器进行编程时，首先将计数常数及定时器方式送入定时器端口（定时器低 8 位及定时器高 6 位、定时器方式）04H 及 05H。计数常数在 0002H~3FFFH 之间选择。

计数器的启动和停止计数由命令寄存器（00H）的最高两位控制。

命令寄存器最高两位（TM<sub>2</sub>、TM<sub>1</sub>）对定时器的控制如下：

- | TM <sub>2</sub> | TM <sub>1</sub> |  |
|-----------------|-----------------|--|
| 0               | 0               | 空操作，不影响计数器操作。  |
| 0               | 1               | 停止计数器计数，当定时器无启动时则无操作。  |
| 1               | 0               | 计数器计满后立即停止计数，若定时器没启动，则无操作。                                   |
| 1               | 1               | 启动，当计数器未计数时，装入计数常数后立即开始计数；若计数器正在计数，则待计数器溢出后按新的工作方式和计数常数开始计数。 |

任何时候都可以设置定时器的长度和工作方式，然后必须将启动命令写入命令寄存器（00H）。即使计数器已经计数，在写入启动命令后仍可改变定时器的工作方式。

如果写入定时器的计数常数为奇数，则方波输出不对称。例如计数常数为 9 时，定时器输出的方波在 5 个脉冲周期内为高电平，4 个脉冲周期内为低电平。

8155 复位后并不预置定时器方式和计数常数。另外，8155 定时器在计数过程中计数器的值并不直接表示外部输入的脉冲数，计数器终值为 2，初值为 2~3FFFH。8155 计数器通常无法用做外部事件计数器，只能用做信号发生器，在输入连续脉冲后，编程输出单方波、连续方波、单脉冲、连续脉冲信号。

**【例 9-2】** 8155 定时器初始化编程。使 8155 用做 I/O 口和定时器工作方式下，A 口定义为基本输入方式，B 口定义为基本输出方式，定时器作为方波发生器，对输入脉冲进行 24 分频（8155 中定时器最高计数频率为 4MHz）。设 I/O 口地址为：

- |          |      |
|----------|------|
| 命令/状态口   | 700H |
| PA 口     | 701H |
| PB 口     | 702H |
| PC 口     | 703H |
| 定时器低 8 位 | 704H |
| 定时器高 8 位 | 705H |

8155 初始化编程如下：

```
MOV DX, 704H      ; 指向定时器低 8 位
MOV AL, 18H        ; 计数常数 0018H=24
```

```

OUT  DX,  AL      ; 计数常数低 8 位装入
INC  DX           ; 指向定时器高 8 位
MOV  AL,  40H     ; 设定定时器方式为连续方波输出
OUT  DX,  AL      ; 定时器高 8 位装入
MOV  DX,  700H    ; 指向命令/状态口
MOV  AL,  0C2H    ; 命令控制字设定 A 口为基本输入方式, B 口为基本输出方式, 并
                  ; 启动定时器

OUT  DX,  AL

```

## 9.3 串行通信及可编程异步通信接口 8250

在计算机应用中, 计算机与外部设备之间、计算机与计算机之间常常要进行数据交换, 我们把这种数据交换称为通信。

数据通信的基本方式有两种, 一种称为并行通信, 另一种称为串行通信。根据并行通信和串行通信的特点, 一般在微机内部或近距离传送数据时, 采用并行通信方式, 而在远距离传送数据时, 采用串行通信方式。

随着数据通信技术的广泛应用和计算机网络技术的不断发展, 在国内、国际上广泛使用了数据通信技术。显然, 无论是计算机内部的通信, 还是计算机与外部设备之间的通信, 或者是计算机之间的通信, 尽管它们的通信方式各有不同, 但通信的双方必须遵守相同的规则, 以达到互相理解。否则, 传送的数据就会成为一堆无用的东西。为使通信能顺利进行, 通信双方要在数据传送方式、双方的同步方式、数据编码、错误校验方式、信息的格式, 以及数据传送速率等方面达成共识, 做出一些基本的规则和约定。这些涉及数据通信的规则和约定, 通常称为通信协议, 或通信规程。

### 9.3.1 串行通信基础

#### 1. 串行通信方式

串行通信中, 数据发送定时和数据接收定时是一个重要问题。串行通信的基本方式分为异步方式和同步方式。

##### (1) 异步通信方式

异步通信的数据链路控制是面向字符的, 即传送的每一组数据构成一个字符, 或者把每个字符看做一个独立的信息进行传送, 并且每个字符出现在数据流中的相对时间是任意的, 而一个字符中每一位占用的时间是固定的, 它由传送速率确定。因此, 异步通信是字符内的同步, 字符间的异步。

异步通信规程规定: 数据流中传送的每个字符必须由起始位 (1 位低电平) 开始, 以停止位结束 (1 位、1.5 位或 2 位高电平), 称为一帧。起始位和停止位称为帧位。

① 起始位: 起始位是连续一位的低电平 (逻辑 0)。起始位仅通知接收器: 由一个字符开始, 而本身不包含关于随后数据的任何信息, 如长度、类型等。接收器只知道应开始计算位数, 直到停止位让其停止计数为止。



② 数据位：在起始位之后，即发送数据位。数据位通常有 4 种配置情况，即有 5 位、6 位、7 位、8 位数据位。异步通信中大多采用 7 位和 8 位数据位结构，要注意：采用 8 位时，一般不用奇偶校验位。采用的数据位个数由接收器和发送器的数据的类型格式来控制。传送的二进制数据一般用编码表示，常用的有美国信息交换标准码 ASCII，这是一种 7 位编码。还有扩展的 BCD 交换码 EBCDIC，这是一种 8 位编码。在异步通信中，通常采用 7 位的 ASCII 码，但是某些资料文件规定，如不采用 8 位数据位就不能正确发送，这时须置成无奇偶校验位的 8 位数据格式。

③ 奇偶校验位：串行通信中，总会发生传输错误，应有检错方法。异步通信采用一位奇偶校验位来检测错误，由于异步通信间歇发送数据，一次只发送一个字符，故异步错误检测可以一个字符一个字符地进行，奇偶校验位就起这个作用。可以进行奇校验或偶校验，由发送端和接收端共同约定。此种检错方法简单，但只能检测一位错误；若有多位出错，尤其是偶数位出错（偶数个位），则此方法失效。

④ 停止位：一帧的最后是停止位，表示一个字符发送结束。停止位可以用软件选择 1 位、1.5 位、2 位高电平。大多数微机通信用一位停止位，因为停止位不含有用信息，它的持续时间可以选择为最小允许值，以尽可能地提高数据传输的速率。

停止位后若不紧跟下一帧信息，这时停止位后一直维持“1”的状态，称这些位为“空闲位”；空闲位的这个“1”状态使下一帧起始位到达时，保证通信线上能引起从“1”到“0”的跳变，从而识别新的一帧。

## (2) 同步通信方式

在异步通信方式中，每一个字符要用起始位和停止位作为字符的开始和结束，这占用了传送时间，至少使传送效率降低 20%，所以在高速数据块传送时，会去掉这些标志，采用同步通信。

在同步通信中，要求发送端总是在正式发送数据之前，先发送一个同步字符通知接收端；接收端在收到同步字符后，便开始按照双方约定的格式和速率接收数据，也就是用同步字符作为信息开始传送的标志和启动信号，接收端没有收到同步字符，便不接收数据，以此实现数据通信的同步。同步通信时，数据连续发送，代码间不留空隙，严格按约定的格式和速率传送，因此适于高速串行传送。

在同步传送时，由于同步字符的不同，可有不同的信息格式，一般分为：

单同步：只有一个同步字符。

双同步：有两个同步字符。

外同步：没有同步字符，靠外部时钟同步。

**SDLC/HDLC：**同步数据链路控制/高级数据链路控制。

同步字符是一种同步标志，指示传送数据的开始。数据是指连续传送的信息，每个字符可选择为 5、6、7、8 位，传送的内容可以是数据信息，也可以是命令信息。CRC 是循环冗余校验，与奇偶校验类似，用于数据传送中的检错。

1969 年由 IBM 公司首先发表了同步数据链路控制规程 SDLC。接着由国际标准化组织 ISO 在 SDLC 基础上提出了高级数据链路控制规程 HDLC，并推荐为国际标准。HDLC 适用于分时系统，以及计算机间的高速数据通信。SDLC 和 HDLC 除了所用的某些术语和技术

细节不同外，其基本原理是相同的。

① 标志场：标志场由固定的 8 比特序列 01111110 组成，表示一帧的开始和结束，也兼作帧同步信号用。为防止在标志场以外的地方，如地址场、控制场等出现同样的序列，发送端和接收端采用“发送 0 比特插入，接收 0 比特删除”技术。具体地说：每当发送器在发送了 5 个连续的“1”后，就自动插入一个“0”，然后继续发送后面的数据；在接收端遇到 5 个连续“1”后的“0”，自动将该“0”删除，这样，可保证标志场的唯一性，而又没有限制发送数据的内容。

② 地址场和控制场：SDLC/HDLC 的一帧信息是由若干场（Field）组成的。前面我们介绍了标志场（F 场），还有地址场（A 场，Address）、控制场（C 场，Control）及数据场（又称信息场，I 场，Information）等。SDLC 规定地址场和控制场的长度为 8 位，而 HDLC 规定地址场为任意长度，控制场的长度为 8 位或 16 位。接收器在接收时检查每个地址字节的第一位，如果为“0”，则后边还跟着一个地址字节；若为“1”，则该字节就是最后一个地址字节。控制场与其类似，若控制字节第一位为“0”，则后面还有第二个控制字节；若为“1”，则只有一个控制字节。

地址场的信息表示链路上从站的地址。控制场的信息是给对方站的工作命令，或对命令的响应，可填入帧的类型、是否可以发送或接收、接收顺序号、发送顺序号及其他控制信息。

③ 数据场：要传送的数据信息，其信息的长度任意。传送时，数据一般连接成 8 位字符结构，当然，不是每一帧信息都必须有数据场。

④ 帧校验序列：每一帧信息有两个字节的帧校验序列，采用 16 位的循环冗余校验码。除了标志场和自动插入的“0”以外，所有的信息都参加 CRC 计算。

采用 SDLC/HDLC 方式传送，数据块可为任意长度，不用指明，因为前后都有标志，并且数据块长度不要求是整数个字符。这种通信要求数据连续发送，不能间断，若有间断，则作异常结束处理，连续发送 8 个“1”（此时不能插入“0”）。

SDLC/HDLC 通信网一般由一个主站和多个从站组成，主站与从站之间可通信，从站与从站之间不直接交换信息。各从站的地址是固定的，由地址场给出。

### （3）异步通信与同步通信的主要区别

时钟要求：同步通信要求发送与接收时钟频率精确相等，异步通信要求发送与接收时钟频率基本相等即可。

控制信息：同步通信要求对整个数据块附加帧信息，用于高速数据链路。异步通信要求对每个数据字符均附加帧信息，用于低速设备，低速传送。

校验方式：同步通信采用 16 位循环冗余校验码，可靠性高。异步通信采用 1 位奇偶校验码，可靠性相对较低。

## 2. 波特率（Baud rate）

波特率表示串行数据的传送速率，它表示每秒钟传送的二进制数的位数，是一个速度衡量单位。定义为：1 波特 = 1 位二进制位/秒。

## 3. 串行通信的数据传送方式

串行通信中，通信双方的数据传送方式有三种：

单工方式：这种方式只有一条通信线，数据只允许按一个固定的方向传送，即一方只能发送数据，而另一方只能接收数据。

半双工方式：这种方式也只有一条通信线，通信双方都具备接收或发送数据的能力，但不允许任何一方在同一时刻既发送数据又接收数据，通信双方只能分时地进行发送或接收数据，或者 A 发送、B 接收，或者 B 发送、A 接收。

全双工方式：全双工方式是一对单工方式，有 2 条通信线，要求通信双方都具有完整的发送能力和接收能力，允许在两个方向上同时传送数据。

#### 4. 信号的调制与解调

在串行通信中，计算机输入/输出的是以二进制数表示的数字信号。若收/发双方距离较远，可借用电话线来进行信息传送。不过计算机中的数字信号有很高的谐波频率，要求传输线的频带很宽，而电话线的频带只有 300Hz~3000Hz，用它来传送数字信号时，会产生严重的畸变。所以在远距离通信时，发送方要利用调制器把数字信号转换成模拟信号，以两种不同频率的正弦波来表示“1”和“0”，并发送到通信线上，这一过程称为调制。接收方利用解调器把收到的模拟信号恢复成数字信号，这一过程称为解调。实现调制与解调的装置称为调制解调器（MODEM）。

按照调制技术，一般有 3 种调制方法，即调频（FM）、调幅（AM）和调相（PM）。它们分别按照数字信号的“0”或“1”状态去改变载波（即音频模拟信号）的频率、幅度和相位，所以在数据通信中又常将调频、调幅和调相分别称为频移键控（FSK）法，幅移键控（ASK）法和相移键控（PSK）法。

### 9.3.2 8250 的内部结构

INS8250 是美国国家半导体公司（National Semiconductor Inc.）生产的产品，使用单一+5V 电源，是具有 40 个引脚的双列直插式芯片。该芯片内部有 10 个寄存器，编程灵活，使用方便。8250 的内部结构框图如图 9-11 所示。它由 7 个部分所组成：数据总线缓冲器、选择和控制逻辑、发送控制电路、接收控制电路、波特率产生电路、调制解调器控制逻辑、中断控制逻辑。

#### 1. 数据总线缓冲器

数据总线缓冲器提供 8250 与 CPU 的数据接口。通过数据总线，8250 可以接收来自于 CPU 的编程控制字及要发送的数据字节，也可以向 CPU 提供接收的数据字节及相应的接收状态，以及调制解调器的状态报告等。

#### 2. 选择和控制逻辑

选择和控制逻辑提供 8250 与 CPU 的地址线和控制线接口。该部分电路接收地址线信号并进行译码，选择芯片内部的各个寄存器。同时控制逻辑也按照 CPU 发出的控制信号在内部产生控制序列，有时结合控制字，对 8250 的各寄存器进行读/写控制。

#### 3. 发送控制电路

发送控制电路由发送保持寄存器 THR、发送移位寄存器 TSR 和发送同步控制电路组成。CPU 要发送的数据首先送入发送保持寄存器 THR；等发送移位寄存器 TSR “空”以后，数据就由 THR 自动送入 TSR；然后再按照编程的各种要求（如数据格式、波特率

等), 加入起始位、奇偶校验位和停止位, 从 8250 的串行数据输出端 SOUT 发送出去。发送的顺序是起始位、数据低位、数据高位、停止位。

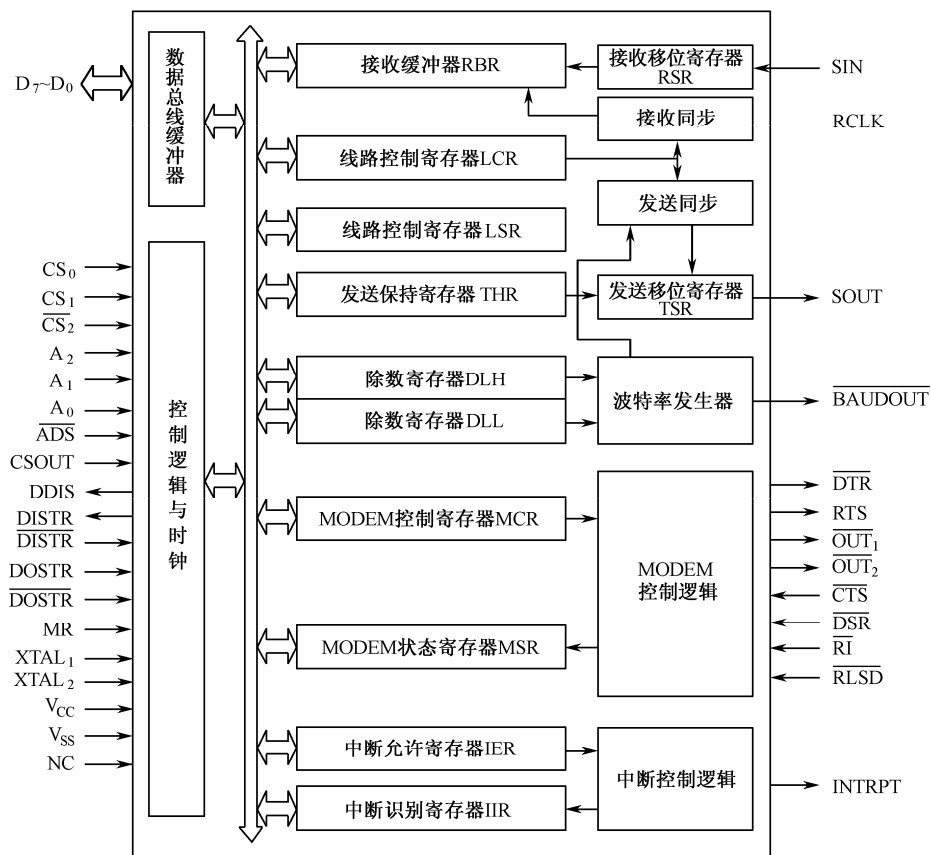


图 9-11 8250 内部结构框图

#### 4. 接收控制电路

接收控制电路由接收移位寄存器 RSR、接收缓冲寄存器 RBR 及接收同步控制电路组成。在接收时, 首先搜索起始位。8250 的数据接收时钟 RCLK 的频率是波特率的 16 倍。接收同步控制电路用 RCLK 的上升沿采样输入信号 SIN, 当采样到 SIN 输入信号由高电平变成低电平, 并连续 8 个 RCLK 周期都是低电平时, 则确认是已经到来的一个起始位; 然后从第 9 个 RCLK 时钟周期的低电平开始算起, 每隔 16 个 RCLK 时钟周期对 SIN 输入的数据进行采样 (此时应是相应数据位的中心), 直至规定的数据格式结束。若正式接收数据前, SIN 上的低电平不能保持连续 9 个 RCLK 时钟周期的话, 则认为是传送线上的干扰所致, 不予理睬, 从头开始搜索 SIN 上的起始位。

#### 5. 波特率产生电路

波特率产生电路由 16 位的除数锁存器 DLH 和 DLL, 以及波特率发生器组成。波特率发生器是一个可编程的分频器, 其分频系数就是除数寄存器的值。根据规定的波特率, 可以计算出应设置的除数寄存器的值:

$$\text{除数寄存器} = \text{基准时钟频率} \div (16 \times \text{波特率})$$

除数寄存器设置后，会产生相应的波特率，也可产生一个工作时钟信号，即：

$$\text{工作时钟频率 } f = 16 \times \text{波特率}$$

这个工作时钟在 8250 芯片内部已作为发送同步控制信号，并经 BOUDOUT 脚输出。

### 6. 调制解调器控制电路

调制解调器控制电路由调制解调器控制寄存器 MCR、调制解调器状态寄存器 MSR 及控制逻辑组成。这部分电路的作用是控制调制解调器的工作。

### 7. 中断控制逻辑

中断控制逻辑由中断允许寄存器 IER、中断识别寄存器 IIR 和中断控制电路组成。8250 支持中断方式传送。8250 内部有多种中断源，它们能否提出申请，以及它们的优先级排序等工作就是由这部分电路来完成和控制的。

### 8. 寄存器组

8250 的寄存器分散在 8250 的七个组成部分之中，但从用户的角度讲，这些寄存器是可直接操作的。8250 共有 10 个寄存器，都是 8 位的，外部用  $A_2A_1A_0$  三位地址线寻址。表 9-3 给出 8250 的 10 个寄存器及其编址，其中 DLAB 是线路控制寄存器 LCR 的  $D_7$  位，用于除数寄存器访问允许。

表 9-3 8250 内部寄存器及其编址

DLAB	$A_2$	$A_1$	$A_0$	寄 存 器
0	0	0	0	接收缓冲器 RBR（读），发送保持寄存器 THR（写）
0	0	0	1	中断允许 IER
×	0	1	0	中断识别 IIR
×	0	1	1	通信线路控制 LCR
×	1	0	0	调制解调器控制 MCR
×	1	0	1	通信线路状态 LSR
×	1	1	0	调制解调器状态 MSR
×	1	1	1	专用
1	0	0	0	除数锁存器 DLL（低 8 位）
1	0	0	1	除数锁存器 DLH（高 8 位）

从功能上看，这 10 个寄存器分为两组，一组用于建立 8250 的基本工作环境，如数据格式、波特率参数、允许中断情况等，这组寄存器包括线路控制寄存器 LCR、调制解调器控制寄存器 MCR、中断允许寄存器 IER、高字节除数锁存器 DLH、低字节除数锁存器 DLL，这 5 个寄存器中的数据通常是在 8250 初始化时写入的，而且一旦写入就很少去改动它们。另一组在 8250 工作中用于实际数据通信，可随时检查各种状态。

### 9.3.3 8250 的引脚功能

8250 的引脚逻辑如图 9-6 所示，可分为两大部分：与 CPU 连接的部分和与通信设备连

接的部分。

### 1. 8250 与CPU连接的引脚

① 数据线  $D_7 \sim D_0$ 。三态，双向，8 位数据线。CPU 通过数据线向 8250 写入控制字及要发送的数据，或者读取 8250 内部寄存器及接收的数据。

② 地址选通  $\overline{ADS}$ 。可用于锁存片选信号和地址信号，低电平有效。若不需要此信号，可将其接地。

③ 数据输入选通  $\overline{DISTR}$ 、 $\overline{DISTR}$ 。用于 CPU 从 8250 读取状态信息或数据，通常只用其中一个即可，另一个可接无效电平。

④ 数据输出选通  $\overline{DOSTR}$ 、 $\overline{DOSTR}$ 。用于 CPU 将数据或控制字写入某一个寄存器，通常只用其中一个即可，另一个可接无效电平。

⑤ 驱动器禁止  $\overline{DDIS}$ 。输出信号，高电平有效。当 CPU 从 8250 读取数据时， $\overline{DDIS}$  变为低电平输出（假若 CPU 与 8250 的数据线上接有驱动器，则此时该驱动器处于允许状态）。当  $\overline{DDIS}$  为高电平时，用于禁止外部驱动器工作。

⑥ 地址线  $A_2 \sim A_0$ 。输入信号，CPU 用这三条地址线寻址 8250 的内部寄存器，如表 9-3 所示。

⑦ 片选信号  $\overline{CS2}$ 、 $\overline{CS1}$ 、 $\overline{CS0}$ 。必须 3 个信号都有效，8250 才能工作。

⑧ 主复位  $\overline{MR}$ 。复位信号，高电平有效。当该信号有效时，8250 进入复位状态，此时除接收缓冲器、发送保持寄存器、除数锁存器外，其余寄存器及控制逻辑均清 0，输出信号均为无效状态，数据输出  $\overline{SOUT}$  呈高电平。 $\overline{MR}$  一般接系统复位信号  $\overline{RESET}$ 。

### 2. 8250 与通信设备连接的引脚

① 基准时钟输入  $\overline{XTAL_1}$ 。8250 的基准时钟输入端，把外部时钟信号或晶振信号接入 8250。

② 基准时钟输出  $\overline{XTAL_2}$ 。输出信号，可用于其他定时控制。

③ 接收时钟  $\overline{RCLK}$ 。接收时钟  $\overline{RCLK}$  用于接收数据时的同步与数据采样，应为波特率的 16 倍频，即： $\overline{RCLK} = 16 \times \text{波特率}$ 。这个时钟信号可由外部电路产生，也可由 8250 自身提供。8250 有一个工作时钟输出信号  $\overline{BAUDOUT}$ ，正好输出 16 倍频波特率的时钟信号，若用此信号，则将  $\overline{RCLK}$  与  $\overline{BAUDOUT}$  两线直接连接即可。

④ 工作时钟输出  $\overline{BAUDOUT}$ 。也叫波特率输出，其频率为 16 倍频波特率。该信号在 8250 内部用于控制发送器的同步，同时也输出到芯片外部。该信号与基准时钟频率、除数寄存器的内容及波特率的关系如下：

工作时钟频率 = 基准时钟频率  $\div$  除数寄存器 =  $16 \times \text{传送波特率}$

该信号也可作为接收时钟，故通常将其连至  $\overline{RCLK}$  端。

⑤ 用户指定的输出  $\overline{OUT_1}$ 、 $\overline{OUT_2}$ 。可以通过对调制解调器控制寄存器  $\overline{MCR}$  的  $D_2$ 、 $D_3$  位编程，使其输出有效的低电平。主复位有效时这两个信号变为高电平。

⑥ 片选输出  $\overline{CSOUT}$ 。是芯片被选中的指示信号，当 8250 被选中时， $\overline{CSOUT}$  变为高电平，此时数据传送才能开始。此信号一般不用。

⑦ 中断请求  $\overline{INTRPT}$ 。高电平有效。当接收出错、接收数据就绪、发送保持器已空或

MODEM 状态改变, 并且中断允许寄存器相应位允许时, 则 INTRPT 变成有效的高电平, 即请求中断。当中断服务结束或主复位后, INTRPT 变为低电平。

⑧ 串行数据输入 SIN。是由通信设备 (外设或 MODEM) 发送的串行输入数据的接收端。

⑨ 串行数据输出 SOUT。这是 8250 发送给通信设备 (外设或 MODEM) 的串行数据的输出端。在主复位后, SOUT 被置成高电平。

⑩ 数据终端就绪  $\overline{\text{DTR}}$ 。低电平有效。有效时表示 8250 通知 MODEM 已做好通信准备。在 8250 收到振铃指示后, 若它已准备好, 则应以此信号回答 MODEM, 此后, 产生振铃的电话线将成为通信的链路。此信号也可由 CPU 使 MODEM 控制寄存器的  $D_0$  位 (DTR 位) 置 “1”, 使它的输出有效。主复位信号使其为高电平。

⑪ 请求发送  $\overline{\text{RTS}}$ 。低电平有效。有效时表示 8250 向 MODEM 发出指示, 它要求向 MODEM 输出串行数据。此信号也可由 CPU 使 MODEM 控制寄存器的  $D_1$  位 (RTS 位) 置 “1”, 使它的输出有效。主复位信号使其为高电平。

⑫ 接收线路信号检测  $\overline{\text{RLSD}}$ 。由 MODEM 提供, 低电平有效。有效时表明 MODEM 已接收到数据载波, 8250 应立即接收解调后的数据。读取 MODEM 状态寄存器的  $D_7$  位可知此信号状态。该寄存器的  $D_3$  位表示上次读取以后, 该位状态是否有变化。若状态中断被允许, 则  $\overline{\text{RLSD}}$  的变化将引起中断。

⑬ 数据设备就绪  $\overline{\text{DSR}}$ 。低电平有效, 是通信设备送给 8250 的控制信号, 表示通信设备就绪, 允许使用通信设备进行数据传送。通信设备一般包括 MODEM、数传机等。通常用 MODEM 泛指通信设备。当 MODEM 已准备好建立通信链路, 准备向 8250 传送数据时, 就向 8250 发出有效的  $\overline{\text{DSR}}$  信号。该位状态可由 MODEM 状态寄存器的  $D_5$  位得到, 此寄存器的  $D_1$  位表示自上次读取以后的变化。若允许该位产生中断, 则  $\overline{\text{DSR}}$  状态的变化就产生中断请求。

⑭ 清除发送  $\overline{\text{CTS}}$ 。低电平有效, 是由 MODEM 送给 8250 的控制信号。有效时表示 MODEM 已同意 8250 的发送请求, 通知 8250 开始发送。它的状态可由读 MODEM 状态寄存器的  $D_4$  位得到。MODEM 状态寄存器的  $D_0$  位指示自上次读取以来  $\overline{\text{CTS}}$  的状态变化。如果允许该状态产生中断, 则  $\overline{\text{CTS}}$  的改变就产生中断请求。 $\overline{\text{RTS}}$  和  $\overline{\text{CTS}}$  是发送数据时使用的一对握手联络信号。

⑮ 振铃指示  $\overline{\text{RI}}$ 。低电平有效。有效时表示 MODEM 已收到电话交换台的拨号呼叫 (电话振铃信号), MODEM 要求 8250 予以回答。此信号状态可由读取 MODEM 状态寄存器的  $D_6$  位 (RI 位) 得到,  $D_2$  位则指出上次读取以后,  $\overline{\text{RI}}$  信号是否改变过, 若该状态中断允许, 则  $\overline{\text{RI}}$  的改变将引起中断。

### 9.3.4 8250 的编程

8250 内部有 10 个可操作的寄存器, 都是 8 位的寄存器。发送保持寄存器 THR 的主要功能是保存发送的数据。接收缓冲寄存器 RBR 的主要功能是保存接收的数据。波特率除数锁存器 DLH 和 DLL 的主要功能是保存对基准时钟频率的分频系数, 以便产生所需的波特率。

当 8250 基准时钟频率为 1.8432MHz（在 PC/XT 中）时，采用分频的办法产生所要求的波特率。8250 接收数据和发送数据所使用的时钟频率是数据传送波特率的 16 倍。当 8250 工作于不同波特率时，就要求有不同的时钟频率，也就要求有不同的分频系数。分频系数即除数，可由下式计算：

除数 = 1843200 / (16 × 波特率)

除数锁存器分为高 8 位 DLH 和低 8 位 DLL，分两次写入。在初始化寻址除数锁存器时，必须先把线路控制寄存器 LCR 的最高位 DLAB 置“1”。在 8250 基准时钟频率为 1.8432MHz 的条件下，8250 的输出波特率与除数锁存器的值之间的关系如表 9-4 所示。8250 的控制字格式如图 9-12 所示。各部分的主要功能为：

线路控制寄存器 LCR：设置一帧数据格式，除数锁存器写入允许位控制，设置附加奇偶位。

线路状态寄存器 LSR：发送与接收是否就绪，接收是否有错误。

调制解调器控制寄存器 MCR：设定自测试方式，控制用户输出信号，控制  $\overline{\text{RTS}}$ 、 $\overline{\text{DTR}}$  信号。

调制解调器状态寄存器 MSR：反映来自 MODEM 的 4 个输入引脚的当前状态和变化状态。

中断识别寄存器 IIR：确定中断源。IIR 是只读的，高 5 位读出时恒为“0”，因此可用这一特征检查某系统中预定的位置是否插入 8250 芯片。

中断允许寄存器 IER：控制 4 种中断类型的允许与禁止。

在使用 8250 之前，初始化编程的主要步骤是：

- ① 80H 写入 LCR（准备写除数）；
- ② 除数写入 DLH，DLL（确定波特率）；
- ③ 写入 LCR，确定数据格式，并必须使  $D_7=0$ ，以便对 RBR、THR、IER 进行操作；
- ④ 写入 IER；
- ⑤ 写入 MCR。

**【例 9-3】** 在 PC/XT 中重新设置 8250。要求：串行异步通信，波特率为 9600 波特，每个字符 7 位数据位，2 个停止位，采用奇校验，允许所有的中断，并用  $\overline{\text{OUT}}_2$  开放 INTRPT 中断。已知 BIOS 为串行接口分配的地址是 3F8H~3FFH。试编写 8250 的初始化程序。

初始化程序如下（有关控制字内容读者可自行猜解）：

表 9-4 波特率与除数锁存器的值之间的关系表

波特率	除数锁存器的值	
	MSB（高）	LSB（低）
50	09	00
75	06	00
110	04	17
134.5	03	59
150	03	00
300	01	80
600	00	C0
1200	00	60
1800	00	40
2000	00	3A
2400	00	30
3600	00	20
4800	00	18
7200	00	10
9600	00	0C



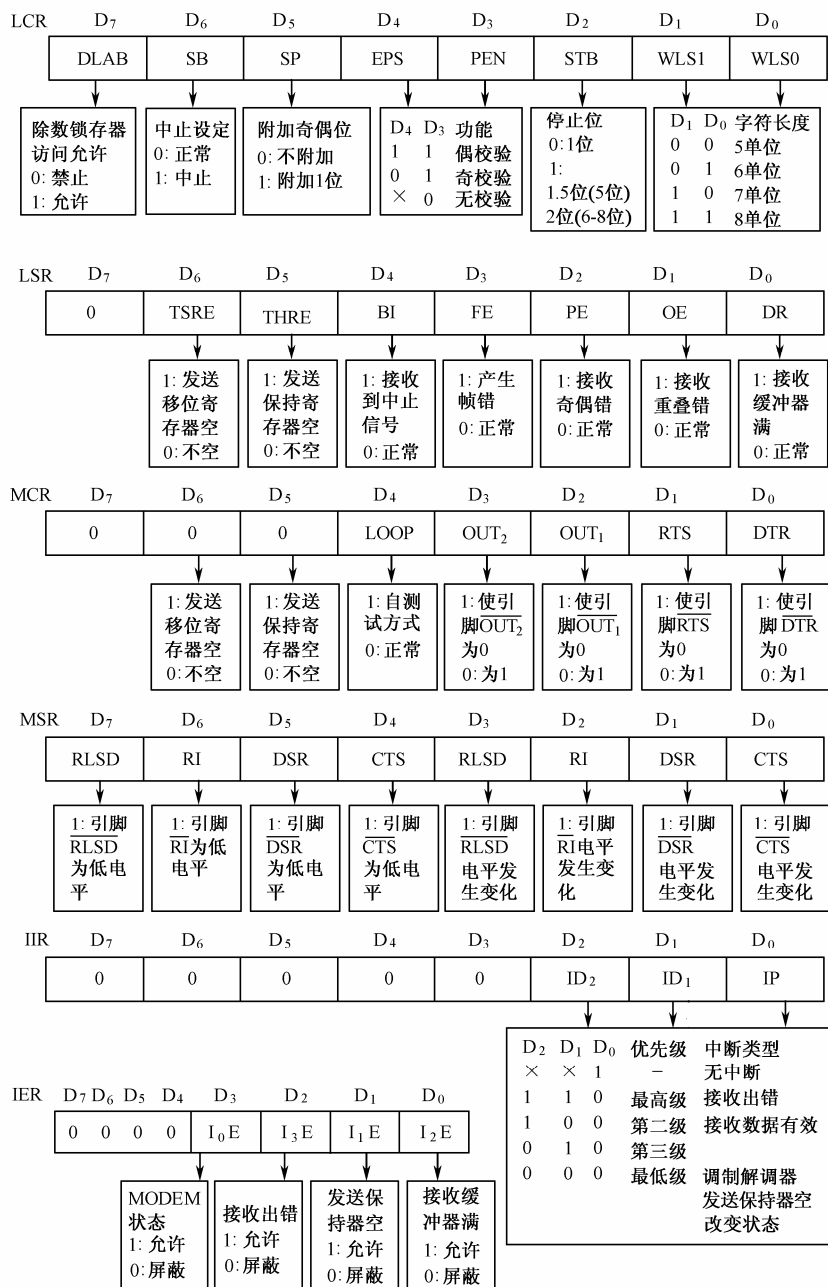


图 9-12 8250 控制字格式

```

MOV AL, 80H
MOV DX, 3FBH
OUT DX, AL          ; 使通信线控制寄存器最高位置 1
MOV AL, 0CH
MOV DX, 3F8H
    
```

```

OUT  DX, AL
MOV  AL, 0
INC  DX
OUT  DX, AL          ; 对除数锁存器置初值
MOV  AL, 00001110B   ; 设置数据格式为 7 位/字符, 两个停止位, 奇校验
MOV  DX, 3FBH
OUT  DX, AL
MOV  AL, 0FH          ; 允许所有中断
MOV  DX, 3F9H
OUT  DX, AL
MOV  AL, 0BH          ; OUT2、DTR、RTS 信号有效
MOV  DX, 3FCH
OUT  DX, AL

```

## 9.4 D/A转换及其接口

### 9.4.1 D/A转换原理

D/A 转换是把数字量信号转换成模拟量信号的过程, 同时, D/A 转换器也是 A/D 转换器的基本组成部分。我们知道: 一个二进制数是由“0”、“1”代码组合起来的, 每位代码都有一定的权。为了将数字量转换成模拟量, 应将每一位代码按权大小转换成相应的模拟输出分量, 然后根据叠加原理将各位代码对应的模拟输出分量相加, 其总和就是与数字量成正比的模拟量, 由此完成 D/A 转换。

D/A 转换的方法较多, 但常用的方法是加权电阻网法和 T 型电阻网法。

#### 1. 加权电阻网法 D/A 转换

加权电阻网法 D/A 转换就是用一个二进制数的每一位代码产生一个与其相应权成正比的电压 (或电流), 然后将这些电压 (或电流) 叠加起来, 就可得到该二进制数所对应的模拟量电压 (或电流) 信号。例如, 让二进制数的第 0 位产生一个  $1V (2^0)$  的电压信号, 第 1 位产生  $2V (2^1)$  的电压信号, 第 2 位产生  $4V (2^2)$  的电压信号, 第 3 位产生  $8V (2^3)$  的电压信号, 依此类推, 第  $n$  位产生  $2^n V$  的电压信号。再将这些电压信号加起来, 就可以得到与原二进制数成正比的电压信号。这种转换方法就称为加权电阻网络法。

#### 2. T型电阻网法 D/A 转换

T 型电阻网法 D/A 转换的组成是:

- ① 输入数据控制的开关组;
- ② R-2R 电阻网络;
- ③ 由运算放大器构成的电流-电压转换电路。

这种转换方法与上述加权电阻网法的主要区别在于电阻求和网络的形式不同, 它采用分流原理来实现对相应数字位的转换。

D/A 转换器的主要性能指标:

#### (1) 分辨率

D/A 转换器的分辨率定义为: 当输入数字发生单位数码变化时, 即 LSB 位产生一次变化时, 所对应输出模拟量(电压或电流)的变化量。实际上, 分辨率反映输出模拟量的最小变化量。对于线性 D/A 转换器来说, 其分辨率与数字量输出的位数  $n$  的关系为:

$$\text{分辨率} = \text{FS}/2^n$$

其中, FS 为模拟输出的满量程值。

#### (2) 精度

D/A 转换器的精度可分为绝对精度和相对精度, 用于表明 D/A 转换的精确程度, 一般用误差大小表示。

① 绝对精度。D/A 转换器的绝对精度(绝对误差)指的是在整个工作区间实际的模拟量输出值与理论输出值之差的最大值。它是由 D/A 转换器的零点调整、增益误差、噪声和线性误差、微分线性误差等引起的。因此, 在手册上常常单独给出各种误差, 以综合说明绝对误差。

② 相对精度。相对精度与绝对精度相似, 不同之处在于, 相对精度把上述的最大偏差表示为满量程输出模拟量的百分数, 或者以二进制位数的形式给出, 例如, 精度为  $\pm 0.1\%$ , 含义是: 最大误差为 FS 的  $\pm 0.1\%$ , 若  $\text{FS}=5\text{V}$ , 则最大误差为  $\pm 5\text{mV}$ 。

### 3. 建立时间

建立时间是描述 D/A 转换速率快慢的一个重要参数, 一般所指的建立时间是指输入数字量变化后, 输出模拟量稳定到相应数值范围内(稳定值  $\pm \varepsilon$ ) 所经历的时间。

### 4. 环境及工作条件

一般情况下, 影响 D/A 转换精度的主要环境和工作条件因素是温度和电源电压变化。D/A 转换器的工作温度按产品等级分为军用级、工业级和商业级, 各级的标准工作温度分别为: 军用级,  $-55^{\circ}\text{C} \sim +125^{\circ}\text{C}$ ; 工业级,  $-25^{\circ}\text{C} \sim +85^{\circ}\text{C}$ ; 商业级,  $0^{\circ}\text{C} \sim -70^{\circ}\text{C}$ 。

## 9.4.2 8 位 D/A 转换器

DAC0832 是美国国家半导体公司推出的 8 位 D/A 转换器。该芯片采用 CMOS 工艺, 双列直插式封装, 可直接与 8080、8088、Z80 等 8 位微处理器以及 MCS-51 系列单片机直接接口, 是在 8 位 D/A 转换器中使用率最高的一种芯片。

### 1. DAC0832 的主要特性

DAC0832 的主要特性有: 8 位分辨率; 电流型输出; 外接参考电压:  $-10\text{V} \sim +10\text{V}$ ; 可采用双缓冲、单缓冲或直接输入三种工作方式; 单电源:  $+5\text{V} \sim +15\text{V}$ ; 电流建立时间:  $1\mu\text{s}$ ; 低功耗:  $20\text{mV}$ ; R-2R T 型解码网络; 线性误差:  $0.2\%\text{FS}$  (FS 为满量程值); 非线性误差:  $0.4\%\text{FS}$ ; 增益温度系数:  $0.002\%\text{FS}/^{\circ}\text{C}$ ; 数字输入与 TTL 兼容。

### 2. DAC0832 的内部构造

DAC0832 由四部分组成: 一个 8 位输入寄存器, 一个 8 位 DAC 寄存器, 一个 8 位 D/A 转换器和一组控制逻辑, 如图 9-13 所示。

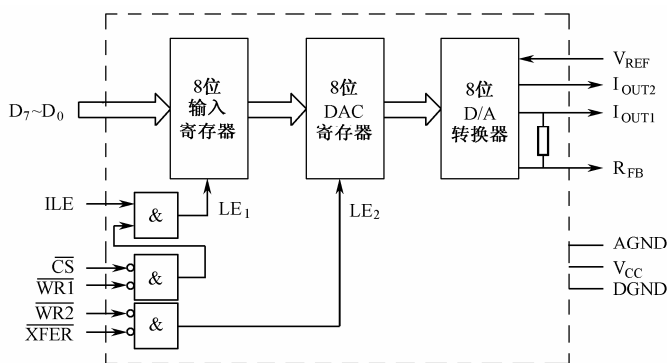


图 9-13 DAC0832 内部结构

在 D/A 转换器中采用的是 T 型 R-2R 电阻网，DAC0832 是电流型输出，改变参考电压  $V_{REF}$  的极性，可以相应地改变输出电流的流向，从而控制输出电压的极性。

8 位输入寄存器受控于控制信号  $ILE$ 、 $\overline{CS}$ 、 $\overline{WR1}$ 。当  $ILE=1$ ， $\overline{CS}=0$ ， $\overline{WR1}=0$  时，该寄存器被选中，允许接收数据线上的信息。当上述控制信号无效时（只要有一个无效即满足条件），锁存该信息。

8 位输入寄存器接收到数据后，并不意味着就进行 D/A 转换。能否真地进行 D/A 转换，还要看数据能否通过 8 位 DAC 寄存器。8 位 DAC 寄存器受控于  $\overline{WR2}$  和  $\overline{XFER}$  信号。当  $\overline{WR2}=0$ ， $\overline{XFER}=0$  时，8 位 DAC 寄存器锁存 8 位输入寄存器的内容。只有 DAC 寄存器的内容才能直接进行 D/A 转换。显然，D/A 转换的输出是无法被禁止的，因为无论如何，DAC 寄存器会有一种数据组合，而这种数据组合将直接进行数字量至模拟量的转换，并直接输出。所以在控制系统的应用中，应在系统初始化时就将 DAC0832 设置成一个安全状态，以避免执行机构的误操作。

DAC0832 在使用时可以采用双缓冲方式、单缓冲方式，或接成直通方式，因此，该芯片使用起来非常方便。

### 3. DAC0832 引脚功能

DAC0832 的引脚配置如下：

$D_7 \sim D_0$ ：8 位数据输入端。

$ILE$ ：输入寄存器允许信号，输入、高电平有效。

$\overline{CS}$ ：片选信号，低电平有效。

$\overline{WR1}$ ：输入寄存器写选通信号，输入、低电平有效。输入寄存器的锁存信号  $LE_1$  由  $ILE$ 、 $\overline{CS}$  和  $\overline{WR1}$  的逻辑组合产生， $LE_1$  为高电平时，输入寄存器状态随输入数据变化， $LE_1$  的负跳变将输入数据锁存。

$\overline{XFER}$ ：传送控制信号，低电平有效。

$\overline{WR2}$ ：DAC 寄存器的写选通信号。DAC 寄存器的锁存信号  $LE_2$  由  $\overline{XFER}$  和  $\overline{WR2}$  的逻辑组合而成。 $LE_2$  为高电平时，DAC 寄存器的输出随寄存器的输入而变化， $LE_2$  负跳变时，输入寄存器的内容打入到 DAC 寄存器并开始 D/A 转换。

$V_{REF}$ ：参考电压，接至内部 T 型电阻网，电压范围为  $-10 \sim +10V$ 。

$I_{OUT1}$ : 电流输出端 1, 其值随 DAC 内容线性变化。当 DAC 寄存器的内容为全“1”时, 输出电流最大; 为全“0”时, 输出电流为 0。

$I_{OUT2}$ : 电流输出端 2。  $I_{OUT1} + I_{OUT2} =$  常数。

$R_{FB}$ : 反馈电阻。由于片内已具有反馈电阻, 故可以与外接运算放大器输出端短接。

$V_{CC}$ : 电源电压, +5V~+15V, 最佳状态是 +15V。

AGND: 模拟地。

DGND: 数字地。

#### 4. DAC0832 的应用

DAC0832 单缓冲方式应用非常广泛, 不但可以对现场执行机构进行控制, 还可用于产生多种周期可调、幅值可变的智能信号。下面编程产生一些波形, 假设 DAC0832 的端口地址为 300H。

##### (1) 方波发生器的程序

```
        MOV    DX, 300H
START:  MOV    AL, 00H
        OUT    DX, AL
        CALL   DELAY1
        MOV    AL, 0FFH
        OUT    DX, AL
        CALL   DELAY1
        JMP     START
```

##### (2) 锯齿波发生器的程序

```
        MOV    DX, AL
        MOV    AL, 00H
        OUT    DX, AL
X1:     INC     AL
        OUT    DX, AL
        CALL   DELAY
        JMP     X1
```

##### (3) 三角波发生器的程序

```
        MOV    DX, 300H
R0:     MOV    CX, 0FFH
        MOV    AL, 00H
R1:     OUT    DX, AL
        INC     AL
        LOOP   R1
        MOV    CX, 0FFH
R2:     DEC     AL
```

```

OUT    DX, AL
LOOP   R2
JMP    R0

```

### 5. DAC0832 的典型连接

DAC0832 的典型连接是单极性单缓冲连接, 如图 9-14 所示。图中 DAC0832 的端口地址为 80H~83H, 使用“OUT 80H, AL”命令将 AL 中的数据输出。

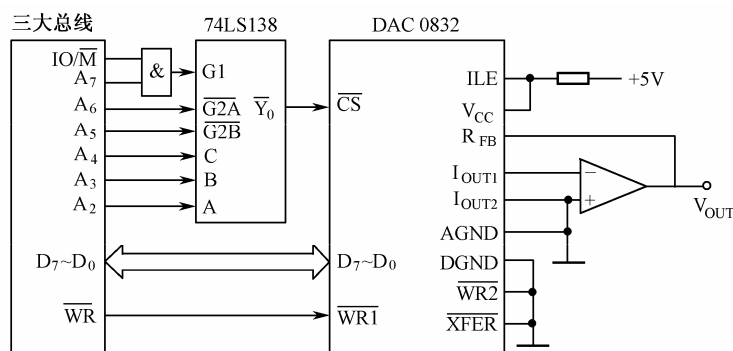


图 9-14 DAC0832 单缓冲接口

【例 9-4】 将从 2000H 开始的 50 个字节单元数据依次送到 DAC0832 输出, 每个数据输出间隔时间为 1ms, 可调用 D1ms 延时 1ms 子程序。

输出程序如下:

```

START: MOV    SI, 2000H
        MOV    CX, 50
X1:     MOV    AL, [SI]
        INC    SI
        OUT    80H, AL
        CALL   D1ms
        LOOP   X1
        HLT

```

### 9.4.3 8 位 CPU 与超过 8 位的 DAC 接口

在许多转换精度及分辨率要求更高的场合, 往往要用大于 8 位的 D/A 转换器, 如 10 位、12 位等。下面以 12 位 D/A 转换器与 8 位 CPU 的接口实例来说明大于 8 位的 D/A 转换器接口的一般方法。

当一个 12 位的 D/A 转换器与 8 位 CPU 接口时, 需把数据宽度分为两段, 即用两个数据锁存器分别锁存低 8 位和高 4 位数据。计算机向 D/A 转换器送数据时分为两次操作, 即先把低 8 位送给低 8 位锁存器, 紧接着把高 4 位送到高 4 位锁存器。

这种方法从接口原理上来说是可以的, 但在实际上却存在问题, 它会在模拟电压输出中出现“毛刺”现象, 这种现象的出现主要是由于两个锁存器的值不同时改变而造成的。

设前一个送入锁存器的数据为： $D_1=0000\ 1111\ 1111$ 。DAC（D/A 转换器）输出电压为  $V_1$ 。下一个送入锁存器的数据为： $D_2=0001\ 0000\ 0000$ 。对应  $D_2$  的输出电压为  $V_2$ 。显然，由于  $D_2 > D_1$ ，因而使  $V_2 > V_1$ 。但当把  $D_2$  的低 8 位送入低 8 位锁存器后，高 4 位送入高 4 位锁存器之前，DAC 的数据锁存器出现了一个中间数据，即 0000 0000 0000，DAC 输出的模拟电压在这一瞬间为 0。在高 4 位数据输出之后，DAC 的数据变为 0001 0000 0000，模拟电压变为正常的  $V_2$ 。这样，输出的模拟电压中出现了一个“毛刺”。

为了解决“毛刺”问题，可采用双缓冲结构，即二重锁存，如图 9-15 所示，使加到 DAC 上的数据同时改变。CPU 先把低 8 位锁存在锁存器 1 中，然后再把高 4 位锁存在锁存器 2 中，最后选中锁存器 3 和锁存器 4，使刚才锁存的 12 位数据同时被传送，对于 DAC 来说，高字节和低字节的数据同时被送到 DAC 的输入端，因此不会产生“毛刺”。

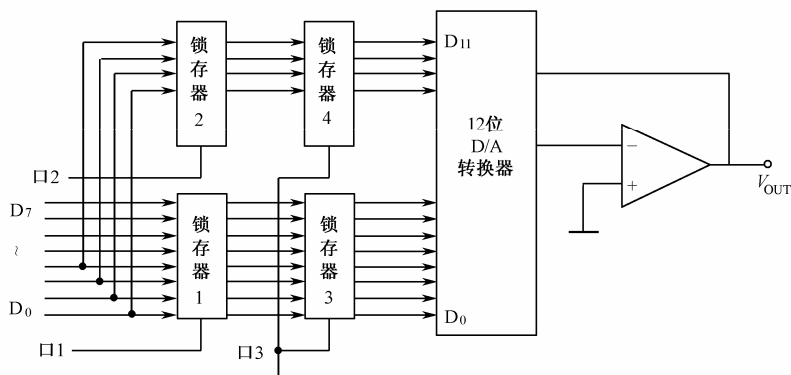


图 9-15 采用二重锁存的 12 位 DAC 接口

实际上，为了节省硬件，只要对高字节采用双缓冲即可。当然，也可对低字节采用双缓冲，总之是节省一个锁存器。

这一方法对于大于 8 位的各种 D/A 转换器适用，但很多大于 8 位的 D/A 转换器已将双重锁存器集成于芯片内部。

#### 9.4.4 12 位 D/A 转换器

DAC1210 芯片属于 DAC1208 系列，是 12 位的 D/A 转换器，有 24 个引脚。

##### 1. DAC1210 的主要特性

DAC1210 的主要特性有：12 位分辨率；电流型输出；外接参考电压： $-10V \sim +10V$ ；可采用双缓冲、单缓冲或直接输入三种工作方式；单电源： $+5V \sim +15V$ ；电流建立时间： $1\mu s$ ；低功耗： $20mW$ ；R-2R T 型解码网络；线性误差： $0.05\%FS$ （FS 为满量程值）；线性误差： $0.05\%FS$ ；数字输入与 TTL 兼容。

##### 2. DAC1210 的内部构造

DAC1210 的内部结构如图 9-16 所示。DAC1210 内部有一个 8 位锁存器，用于锁存高 8 位；有一个 4 位锁存器，用于锁存低 4 位；还有一个 12 位锁存器，用于同时提供 12 位 D/A 转换所需的时间。这几个锁存器分别由 LE1、LE2、LE3 控制，其值为 1 时跟随，为

0 时锁存。

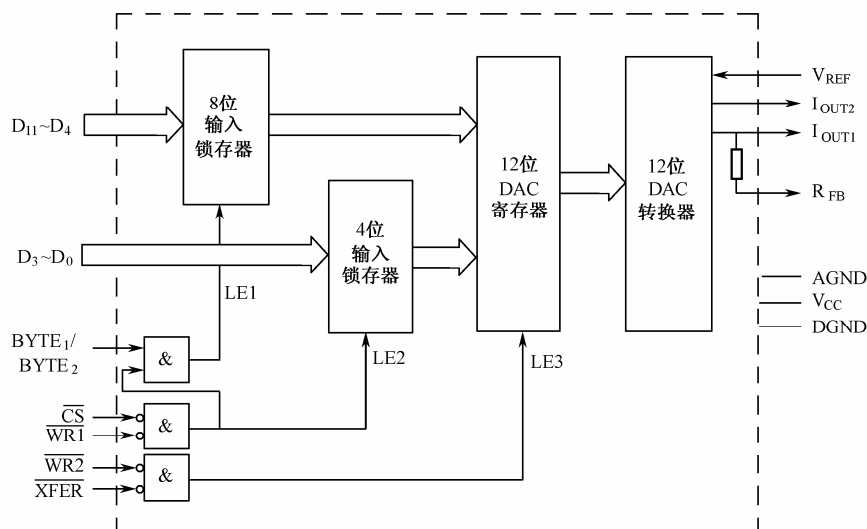


图 9-16 DAC1210 的内部结构

可以看到，DAC1210 在原理、结构、引脚定义和使用上基本与 DAC0832 类似，可以参照。不同之处在于分辨率。

## 9.5 A/D转换及其接口

### 9.5.1 A/D转换的基本过程及转换原理

#### 1. A/D转换的基本过程

模拟量是时间上和幅值上都连续的一种信号。模拟量经过采样后得到的信号是时间上离散、幅值上连续的信号，即离散信号，这一过程就是采样过程。计算机只能处理数字量，所以还必须把离散信号在幅值上也进一步离散化，这一过程就是量化过程。量化后的信号是时间上和幅值上都离散的数字量，可以直接送到计算机中进行处理。

采样是将模拟量变换为离散量，一般包括采样与保持两个过程；量化是将离散量变换成数字量，一般包括量化与编码两个过程。采样与量化是 A/D 转换的基本过程。

##### (1) 采样

为了把一个连续变化的模拟信号转变成对应的数字信号，就必须首先把模拟信号在时间上离散化，也就是对模拟信号进行采样。采样的过程是：先使用一个采集电路，按等距离时间间隔，对模拟信号进行采样，然后用保持电路将采集来的信号电平保持一段时间，以使模/数转换器能正确地将其转换成对应的数字量。

如图 9-17 所示，对输入模拟信号  $f(t)$  按等时间间隔  $T$  进行采样保持，即可得到一个新的阶梯函数  $f(t_n)$ ，其中， $t_n = nT$  ( $n=0, 1, 2, \dots$  为正整数)。 $t_n$  表示采样的时间变量，这是一个



离散的量, 但  $f(t_n)$  (阶梯函数) 的值却是一个连续的量。所谓连续的量并不是说相邻两个阶梯函数值的中间没有断点, 而是指任一个阶梯函数值本身都可以是输入模拟信号最大值和最小值之间的任何一个数值。显然这种数值是连续的。

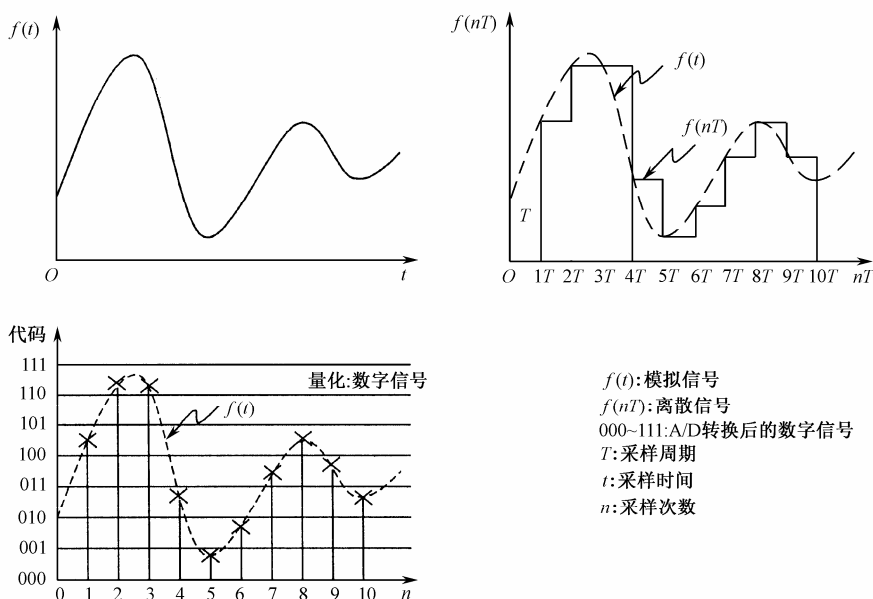


图 9-17 信号的采样和量化

我们把等距离时间间隔  $T$  叫做采样周期, 把  $1/T$  称为采样频率  $f$ , 即  $f=1/T$ 。显然, 采样频率越高, 即  $T$  越小, 则阶梯函数  $f(t_n)$  就越接近于连续的模拟输入信号  $f(t)$ 。但是, 由于实际电子电路的限制, 不可能在很高的频率下对输入模拟信号进行采样, 为此, 就会提出一个问题: 选取什么样的最低采样频率, 可以使得到的离散信号包含了输入模拟信号的主要信息, 并能通过合适的滤波器恢复原来的模拟信号?

这是模/数转换技术必须首先解决的一个理论问题。这个问题换个方式问, 就是: 经过采样以后, 我们不是取全部时间上的信号值, 而只取某些时间点上的值, 这样处理后, 会不会造成信息的丢失呢?

香农 (Shannon) 采样定理定量地回答了这个问题。

**采样定理:** 对一个有限频率谱 ( $\omega < +\omega_{\max}$ ) 的连续信号进行采样, 当采样频率  $f \geq 2f_{\max}$  时 ( $f_{\max}$  是输入模拟信号的最高频率), 则采样输出信号能无失真地恢复原来的连续信号。

采样定理是实现无信息失真地重现采样数据的必要条件。要求对原始数据的采样及数据重构都是理想状态。但在实际使用上, 不可能具有这样的理想情况。为了保证数据采集的精度, 在模拟输入信道中通常采取如下方法:

① 增加每个周期的采样数, 通常根据数据带宽, 在最高频率端每周周期采样 7~10 次, 即  $f = (7 \sim 10)f_{\max}$ 。

② 在 A/D 转换前设置低通滤波器, 消除信号中无用的高频分量。

对于多通道数据采集系统, 由于是分时分路采样的, 因此考虑到通道的分时、模拟信

号的带宽等因素,多通道数据采集系统的最小采样频率应为:  $f = (7 \sim 10)f_{\max} \times N$ , 其中  $N$  为通道数。最大采样周期  $T = 1/f$ 。

## (2) 量化

模拟量输入信号被采样以后,得到的是时间上离散、幅值上连续的信号,即离散信号,但是要想用计算机处理,就必须把这种信号转换为时间上和幅值上都离散的信号,即数字信号。这种把离散信号转变为数字信号的过程,就是量化过程。

如果把每一采样值都看做是实数轴上的任一值,那么量化过程就是用一组有限实数来表示每一采样值。量化过程也很像对一个物体的称重过程,如果我们把每一采样值看做是被称的重物,那么量化过程就可以看做是用一组大小法码来近似表示的这个重物的重量值。所以说是“近似表示”,是因为用数字量表示一个模拟量。我们日常买菜或买食品,一般是精确到以“两”为单位;而在煤矿,精确单位最低也是“斤”,一般是“吨”,从不用“两”;但是在化学实验室,一般的精确单位是“钱”或“克”。所以,一个测量值的精度取决于测量的最小单位。同样,在对采样值的量化过程中,也存在精度问题,那就是用几位二进制数(或BCD码)来表示采样值。

量化就是把输入模拟信号  $f(t)$  的变化范围划分成若干层,每一层都由一个数字来代表,采样值落到哪一层,就由哪一层的数字来代表。这样,所有的采样值经过“量化”后,就变为对应的数字量,成为整数值。

在量化过程中,量化结果把“零头”按四舍五入法消去了。很明显,量化过程中“化零为整”要产生误差,这种舍入误差是量化过程中的固有误差,最大偏差等于量化单位  $R$  的一半。这种误差不可能消除,只能降低,当量化单位取得愈小时,即  $R$  愈小,误差愈小。

量化以后,量化部件将输出对应的二进制数(或BCD码),并送给计算机;计算机对这种与模拟量相对应的数字量进行处理。

## 2. A/D转换原理

A/D转换是将模拟信号转换成数字信号的过程。A/D转换的方法较多,有计数式、逐次逼近式、双积分式,以及并行转换式等。其中计数式最简单,但转换速度很慢;并行转换式的转换速度最快,但需要的器件多,价格高;逐次逼近式A/D转换器的速度较快,比较简单,而且价格适中,可以说,其各种指标比较适中,因此是微型计算机应用系统中最常用的外围接口电路;双积分式A/D转换器精度高,抗干扰能力强,但速度慢,一般应用在要求精度高而速度不快的场合,例如仪器仪表等。

下面分别介绍计数式、逐次逼近式、双积分式、并行式A/D转换器的工作原理。

### (1) 计数式A/D转换器

计数式A/D转换器的工作原理是:被测电压不断地和一个均匀增长的斜坡电压作比较,直到二者相等时,比较过程才结束,并输出一个代表被测电压值的二进制数。

### (2) 逐次逼近式A/D转换器

逐次逼近式A/D转换器的工作原理也是将被测电压和由D/A转换生成的电压进行比较,但这里D/A转换生成的电压不是以线性增长的方式接近于被测电压,而是用对分搜索的方式来逐次逼近被测电压的。

图 9-18 所示为逐次逼近式 A/D 转换器的工作原理，该电路内部由 4 个部分组成：逐次逼近寄存器 SAR、D/A 转换器、电压比较器和置数选择逻辑电路。

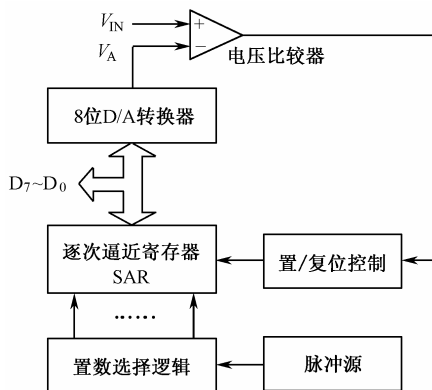


图 9-18 逐次逼近式 A/D 转换器的工作原理

$V_{IN}$  为待转换的模拟电压。转换开始后，首先由控制逻辑电路将逐次逼近寄存器 SAR 的最高位置“1”，其余位置“0”，然后再将该假定数据送 D/A 转换器，变成模拟电压  $V_A$ 。将  $V_A$  与输入的模拟电压信号  $V_{IN}$  一起送入电压比较器进行比较，如果  $V_{IN} < V_A$ ，则说明将 SAR 寄存器的最高位置“1”不合适，应置成“0”；否则，说明将 SAR 寄存器的最高位置“1”合适，应保留下来。然后，再将 SAR 寄存器的次高位置“1”，重复上述的转换、比较、判断，以决定该位置“1”还是置

“0”。上述过程反复进行，直到确定了 SAR 寄存器的最低位为止。这样 SAR 寄存器中的内容就是  $V_{IN}$  所转换成的二进制数。最后将 SAR 寄存器中的数据送入输出缓冲器，准备输出。

### （3）双积分式 A/D 转换器

双积分式 A/D 转换器的基本原理是，将一段时间内的模拟电压通过两次积分，转换成与模拟电压成正比的时间间隔，然后利用时钟脉冲和计数器测出此段时间间隔，所得到的计数结果就是输入电压对应的数字值。

双积分式 A/D 转换器的工作原理框图如图 9-19 所示，它是由积分器、过零比较器、计数器和控制电路组成的。

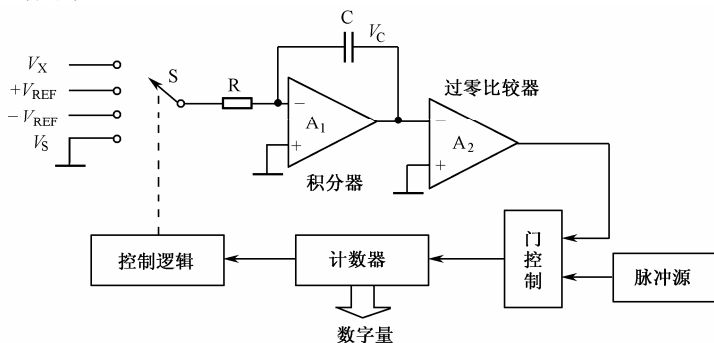


图 9-19 双积分式 A/D 转换器的工作原理框图

双积分式 A/D 转换器的工作步骤如下：

第一步，初始化阶段。将开关 S 接到  $V_S$ （接地）端，使积分器处于原始状态， $V_C = 0$ 。

第二步，采样阶段。将 S 接到模拟量输入端  $V_X$ ，在  $T_1$ （规定的时间）时间内积分。此时，积分器从原始状态开始积分。当积分到时间  $T_1$  时，积分器的输出为：

$$V_C = -\frac{1}{RC} \int_0^{T_1} V_X dt = -\frac{T_1}{RC} \frac{1}{T_1} \int_0^{T_1} V_X dt = -\frac{T_1}{RC} \overline{V_X}$$

式中， $\overline{V_X} = \frac{1}{T_1} \int_0^{T_1} V_X dt$  是输入电压在  $T_1$  时间内的平均值。

可见, 经过第一次定时积分后, 积分器的输出  $V_C$  与输入电压的平均值  $\overline{V_X}$  成正比。令此时的积分器输出为  $V_A$ , 即

$$V_A = V_C = -\frac{1}{RC} T_1 \overline{V_X} \quad (9-1)$$

第三步, 比较阶段。将  $S$  接到与  $V_X$  极性相反的基准电源  $-V_{REF}$  上, 此时, 积分器就要由  $V_A$  开始反向积分到 0, 这段时间用  $T_2$  表示。此时积分器的输出为:

$$V_C = V_A - \frac{1}{RC} \int_0^{T_2} (-V_{REF}) dt = 0$$

$$\text{即} \quad V_A = -\frac{1}{RC} T_2 V_{REF} \quad (9-2)$$

由式 (9-1) 及式 (9-2) 可得:

$$-\frac{1}{RC} T_1 \overline{V_X} = -\frac{1}{RC} T_2 V_{REF}$$

即  $T_2 = \frac{T_1}{V_{REF}} \overline{V_X}$ 。式中  $T_1$  为规定的时间间隔,  $V_{REF}$  为固定的基准电源。由此可得出结论: 积分时间间隔  $T_2$  与输入电压在  $T_1$  时间内的平均值  $\overline{V_X}$  成正比。

双积分式 A/D 转换器的工作过程简述如下:

一开始,  $S$  接地,  $V_C=0$ , 使积分器处于原始状态, 紧接着  $S$  接  $V_X$ , 于是积分器对  $V_X$  进行积分, 积分器的输出电压  $V_C$  从 0 开始下降。当积分器对  $V_X$  积分到  $T_1$  时间时, 采样结束,  $S$  转接  $-V_{REF}$ , 积分器开始对基准电源电压反向积分, 积分器输出电压波形从负值  $V_m$  以固定斜率往正方向回升。如果我们将开关  $S$  接基准电源电压  $-V_{REF}$ , 积分器开始反向积分的  $t_2$  时刻, 打开计数控制门, 计数器开始计数; 当积分器反向积分到 0 的  $t_3$  时刻, 检零比较器输出信号, 使控制门关闭, 停止计数, 那么这个计数结果就是对反向积分时间  $T_2$  的计数, 也就是输入电压在  $T_1$  时间内平均值  $\overline{V_X}$  的数字量。

综合上述分析说明, 计数器所记录的数值与输入电压在  $T_1$  时间内的平均值  $\overline{V_X}$  成正比, 即  $\overline{V_X}$  越小,  $V_m$  就越小, 于是时间间隔  $T_2$  就越短, 进而计数的脉冲个数也就越少。

### 3. A/D 转换器的主要性能指标

#### (1) 分辨率

A/D 转换器的分辨率表述为: 输出数字量变化一个相邻数码所需输入模拟电压的变化量, 亦即与芯片最低有效位 (LSB) 相当的模拟电压的值。A/D 转换器的分辨率习惯上以输出二进制数的位数或 BCD 码的位数来表示。如果一个 A/D 转换器输出二进制数的位数为  $n$ , 则:

$$\text{分辨率} = FS/2^n \quad (FS \text{ 是输入电压满量程值})$$

#### (2) 精度

精度是指 A/D 转换器输出的数字量所对应的实际输入电压值与理论上产生该数字量的应有的输入电压之差, 它反映实际 A/D 转换器与理想 A/D 转换器的差别, 常用误差表示。精度分为绝对精度和相对精度。

#### (3) 转换时间

转换时间是指完成一次 A/D 转换所需要的时间, 即从启动 A/D 转换器开始工作, 到转

转换结束所经历的时间间隔。转换时间的倒数称为转换速率。例如，ADC0809 的转换时间为  $100\mu\text{s}$ ，则转换速率为每秒 1 万次。

### 9.5.2 8 位A/D转换器

A/D 转换器产品种类繁多，这些产品在字长、速率、隔离状态、多路分时采集方式方面各有不同。本节以 ADC0809 为例，介绍 A/D 转换芯片。

ADC0809 是美国 NS 公司生产的 CMOS 组件，为 8 路输入单片 A/D 转换器，可直接与 CPU 总线连接，使用非常广泛。

#### 1. ADC0809 主要特性

- 8 位分辨率；
- 电压输入： $0\sim+5\text{V}$ ；
- 转换时间： $100\mu\text{s}$ （640kHz 条件）；
- 时钟频率： $100\sim1280\text{kHz}$ ，标准时钟为 640kHz；
- 无漏码；
- 单一电源： $+5\text{V}$ ；
- 8 路单端模拟量输入通道；
- 参考电压： $+5\text{V}$ ；
- 总的不可调误差： $\pm 1\text{LSB}$ ；
- 温度范围： $-40^{\circ}\text{C}\sim+85^{\circ}\text{C}$ ；
- 功耗低： $15\text{mW}$ ；
- 不需进行零点调整和满量程调整；
- 可锁存的三态输出；
- 输出与 TTL 电路兼容。

#### 2. ADC0809 内部结构

ADC0809 是一种 CMOS 单片 A/D 转换器，其内部结构如图 9-20 所示，由三个部分组成，即 8 路模拟开关及地址锁存与译码、8 位 A/D 转换器、三态输出锁存缓冲器。

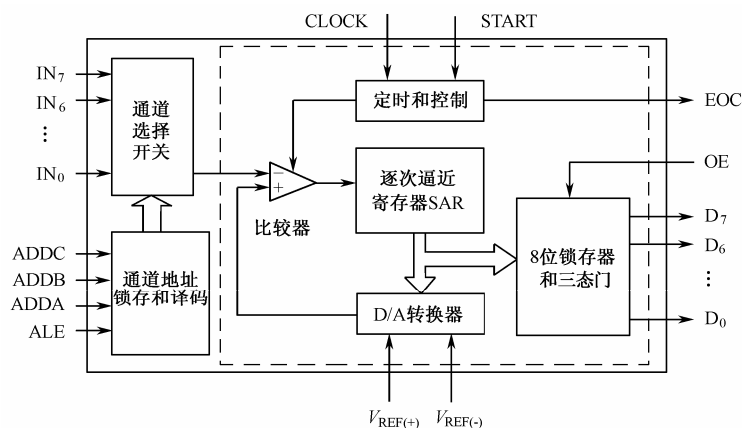


图 9-20 ADC0809 内部结构

ADC0809 中包含 8 个标准的模拟开关, 8 个输入模拟量可以通过引线  $IN_0 \sim IN_7$  输入。多路开关的状态由三位地址信号译码控制, 某一时刻只能有一路模拟信号进行 A/D 转换。地址信号与通道选择的对应关系如表 9-5 所示。

表 9-5 8 路模拟输入通道与地址选择

地址 ADDR			模拟量 输入通道
ADDC	ADDB	ADDA	
0	0	0	$IN_0$
0	0	1	$IN_1$
0	1	0	$IN_2$
0	1	1	$IN_3$
1	0	0	$IN_4$
1	0	1	$IN_5$
1	1	0	$IN_6$
1	1	1	$IN_7$

ADC0809 采用逐次逼近式转换方法。该转换器包括比较器、逐次逼近寄存器 SAR、开关树、256R 网络和控制逻辑等部件。这些部件的作用和逐次逼近式的转换原理, 前面已有叙述, 这里就不重复了。其中开关树是一个接受逐次逼近寄存器控制的开关阵, 开关树中各开关的状态, 通过接通或断开 256R 网络中的某些支路, 从标准参考电压逐次得到对应的推测值, 送往比较器的输入端, 和输入的模拟量进行比较。

当 8 位数据按位比较后, 得到一个确定的数字量。该数字量在三态输出锁存器中锁存, 并经三态缓冲器与计算机数据总线连接, CPU 读取转换结果。

ADC0809 内部没有模拟输入信号采样保持器, 处理快速信号时应外加。

### 3. ADC0809 引脚功能

ADC0809 的引脚排列如图 9-21 所示。各引脚功能如下:

$IN_7 \sim IN_0$ : 8 路模拟量输入线。

$D_7 \sim D_0$ : 8 位数字量输出线。

ADDC $\sim$ ADDA: 3 位地址线、用来选通 8 路模拟通道中的 1 路。

ALE: 地址锁存允许。在 ALE 的上升沿, ADDC、ADDB、ADDA 三位地址信号被锁存到地址锁存器。

START: 启动信号, 正脉冲有效。地址锁存后, 在该引脚加一正脉冲, 该脉冲的上升沿使所有内部寄存器清 0, 其中包括使逐次逼近寄存器复位, 从下降沿开始进行 A/D 转换。如果正在进行转换时, 接到新的启动信号, 则原来的转换进程被中止。

CLOCK: 时钟输入信号。其时钟频率范围是 100 $\sim$ 1280kHz, 标准时钟为 640kHz, 转

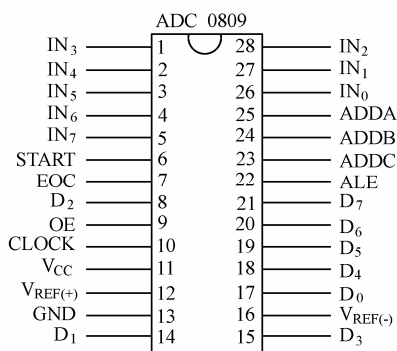


图 9-21 ADC0809 引脚排列

换时间为  $100\mu\text{s}$ 。据测量，当时钟频率为  $500\text{kHz}$  时，转换时间为  $128\mu\text{s}$ ；当时钟频率为  $1\text{MHz}$  时，转换时间为  $66\mu\text{s}$ 。

**EOC：**转换结束输出信号。当该信号为低电平时，表明 ADC0809 已准备开始转换，或转换正在进行之中，不能提供一个有效的稳定的数据。在 START 的上升沿清 0 ADC0809 的内部寄存器，准备开始转换，此时 EOC 变为低电平。在 START 的下降沿启动 A/D 转换器，开始转换，此时 EOC 仍为低电平。当转换结束后，转换数据已锁存到输出锁存器时，EOC 变为高电平。当 EOC 变为高电平时，表示转换已经结束，A/D 转换器可提供有效数据。EOC 可作为被查询的状态信号，亦可用于申请中断。ADC0809 转换一次共需 64 个时钟周期（CLOCK 周期）。ADC0809 的工作时序如图 9-17 所示。

**OE：**输出允许，输入线。当 OE 为高电平时打开输出三态缓冲器，使转换后的数据进入数据总线。

$V_{\text{REF}(+)}$ 、 $V_{\text{REF}(-)}$ ：基准电压输入。一般应用情况下， $V_{\text{REF}(+)}$  接  $+5\text{V}$ ，而  $V_{\text{REF}(-)}$  与 GND 相连。

$V_{\text{CC}}$ ：电源电压，接  $+5\text{V}$ 。

GND：地信号。

ADC0809 的模拟量输入是单极性的，范围为  $0\sim+5\text{V}$ 。若实际模拟输入信号是双极性的，比如为  $-5\text{V}\sim+5\text{V}$ ，则需要设计极性转换电路。图 9-18 所示为常用的极性转换电路，若信号源的内阻较小，可采用如图 9-22(a)所示的电路；若信号源的内阻较大，可采用如图 9-18(b)所示的电路。

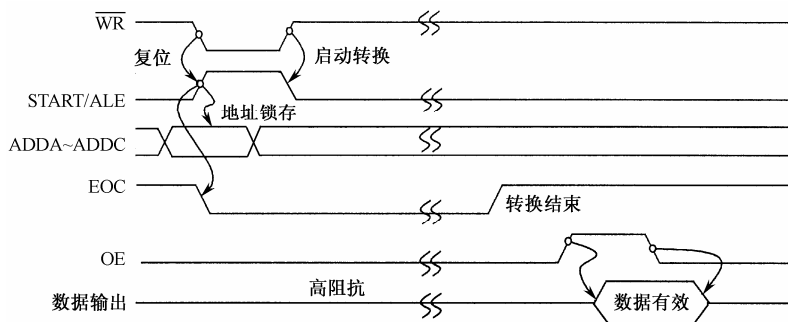


图 9-17 ADC0809 工作时序

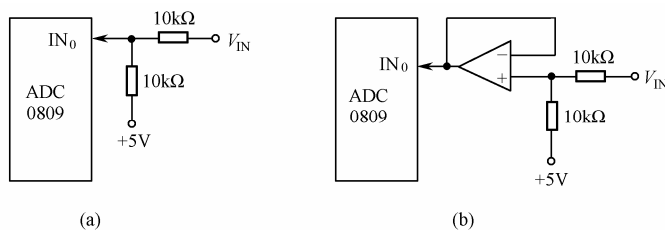


图 9-22 ADC0809 常用的极性转换输入电路

4. ADC0809 与 CPU 的接口方法

A/D 转换器与 CPU 的数据传送控制方式通常有以下 3 种。

(1) 等待方式

等待方式又称定时采样方式，或无条件传送方式，这种方式是在向 A/D 转换器发出启动指令（脉冲）后，进行软件延时（等待），延时时间取决于 A/D 转换器完成 A/D 转换所需要的时间（如 ADC0809 在 640kHz 时为 100μs），经过延时后才可读入 A/D 转换数据。在这种方式中，有时为了确保转换完成，不得不把延时时间适当延长，因此，比查询方式的转换速度慢，但对硬件接口要求较低，可视系统 CPU 紧张程度选用。

通常在 CPU 非常空闲（无事可干，只等采样）的情况下，还是采用等待方式为好，以节约系统成本，减少硬件设计的工作量，提高系统可靠性。

等待方式下，ADC0809 与微处理器之间的连接如图 9-23 所示。图中译码器的输出作为 ADC0809 的转换启动地址 START（同时通道地址锁存信号 ALE 有效）和数字量数据输出地址 OE，转换结束信号 EOC 未用。若采集通道 IN<sub>0</sub> 的数据，可设计如下程序：

```
MOV    AL, 00H      ; 设置通道号 0
OUT    84H, AL      ; 启动 0 通道进行 A/D 转换
CALL   DELAY100     ; 延时 100μs，等待 A/D 转换结束
IN     AL, 84H      ; 转换结束，读入 A/D 转换结果
```

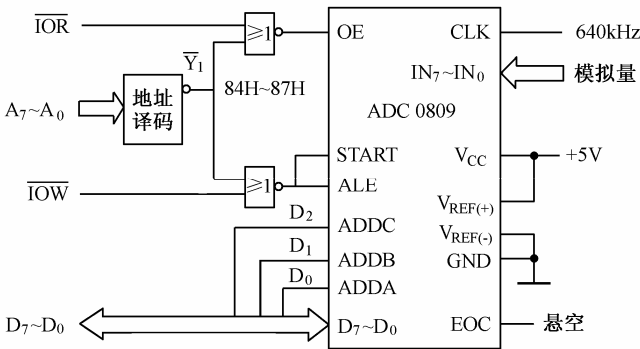


图 9-23 等待方式下 ADC0809 与微处理器之间的连接

(2) 查询方式

所谓程序查询方式，就是先选通模拟量输入通道，发出启动 A/D 转换器的信号，然后用程序查看 EOC 状态：若 EOC=1，则表示 A/D 转换已结束，可以读入数据；若 EOC=0，则说明 A/D 转换器正在转换过程中，应继续查询，直到 EOC=1 为止。

查询方式下，ADC0809 与微处理器之间的连接如图 9-24 所示。利用该接口电路，采用查询方式，对现场 8 路模拟量输入信号循环采集一次，其数据存入数据缓冲区中。程序设计如下：

```
DATA    SEGMENT
COUNT  DB    00H      ; 采样次数
NUMBER  DB    00H      ; 通道号
```



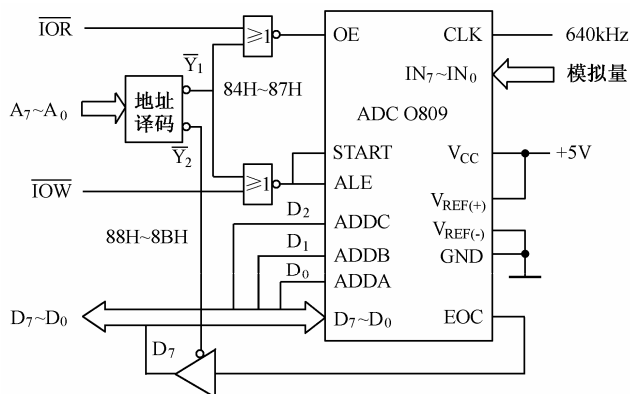


图 9-24 查询方式下，ADC0809 与微处理器之间的连接

```

ADCBUF DB 8 DUP(?) ; 采样数据缓冲区
DATA   ENDS
ADCC EQU 84H ; A/D 控制口地址
ADCS EQU 88H ; A/D 状态口地址
CODE   SEGMENT
        ASSUME CS:CODE,DS:DATA
START:  MOV AX, DATA
        MOV DS, AX
        MOV BX, OFFSET ADCBUF ; 设置 A/D 缓冲区
        MOV CL, COUNT ; 设置采样次数
        MOV DL, NUMBER ; 设置通道号
X3:     MOV AL, DL
        OUT ADDC, AL ; 启动 ADC0809 相应通道
X1:     IN AL, ADCS ; 读取状态口
        TEST AL, 80H ; 析取 EOC
        JNZ X1 ; EOC≠0, ADC0809 未开始转换, 等待
X2:     IN AL, ADCS
        TEST AL, 80H
        JZ X2 ; EOC≠1, ADC0809 未转换完成, 等待
        IN AL, ADCC ; 读数据
        MOV [BX], AL
        INC BX ; 指向下一个数据缓冲单元
        INC DL ; 指向下一个通道
        INC CL ; 采样次数加 1
        CMP CL, 08H
        JNZ X3
        MOV AX, 4C00H
    
```

```

                INT    21H
CODE           ENDS
                END    START

```

这种方法程序设计比较简单，且可靠性比较高，但实时性差，把 CPU 的大量时间都消耗在“查询”上了（当然，比等待方式速度快）。因此，这种方法只用在实时性要求不太高，或者控制回路比较少的控制系统。而大多数控制系统对于这点时间是允许的，因此，这种方法也是用得最多的一种方式。

### （3）中断方式

在前两种方式中，无论 CPU 暂停与否，实际上对控制过程来说都是处于等待状态，等待 A/D 转换结束后再读入数据，因此速度慢。为了发挥计算机的效率，有时采用中断方式。在这种方式中，CPU 启动 A/D 转换器后，即可转去处理其他事情，比如继续执行主程序的其他任务。一旦 A/D 转换结束，则由 A/D 转换器发出转换结束信号，这一信号作为中断请求信号送给 CPU，CPU 响应中断后，便读入数据。这样，在整个系统中，CPU 与 A/D 转换器是并行工作的，提高了系统的工作效率。

中断方式不需要等待时间，但若中断后，保护现场、恢复现场等一系列操作过于烦琐，所占用的时间和 A/D 转换的时间相当，则中断方式就失去了它的优越性。

对于 ADC0809，除非它正处于 A/D 转换过程中，否则它的 EOC 就为高电平。对于有些 CPU 来说，高电平意味着申请中断（比如 8088 CPU 的 INTR）。为了保证 ADC0809 确实是在转换完成后才产生中断，而且仅仅是产生一次中断，应重新设计一个中断逻辑电路。当然，如果系统中没有其他的中断源，也可以采用软件的方法解决这个问题，其方法是：先启动 ADC0809，延迟一小段时间后开中断，然后执行其他程序。当 CPU 响应中断后，系统自动关中断，在下一次启动前不再开放，以保证每一次 A/D 转换后只响应一次。下一次 A/D 转换依次循环。

## 9.5.3 12 位 A/D 转换器

AD574 型快速 12 位逐次逼近式 A/D 转换器是美国 AD 公司的产品，是一种内部由双极型电路组成的 28 脚双列直插式标准封装的集成 A/D 转换器，该转换器集成度高，并因其功能完善和性能优异而被广泛采用。

### 1. AD574 主要特性

- 12 位分辨率。
- 转换时间：25 $\mu$ s（12 位），16 $\mu$ s（8 位）。
- 单通道模拟电压输入，无漏码，采用逐次逼近式原理。
- 电压输入：单极性，0~10V，0~20V；双极性， $\pm 5$ V， $\pm 10$ V。
- 供电电源：V<sub>LOGIC</sub> 逻辑电平：+4.5V~+5.5V（+5V）；  
V<sub>CC</sub>：+13.5V~+16.5V（+12V，+15V）；  
V<sub>EE</sub> 供电电源：-13.5V~-16.5V（-12V，-15V）。
- 参考电压不需外部提供，芯片内部具有稳压值为 10.00V $\pm$ 0.1V(max)的参考电压。

- 片内具有输出三态缓冲器，可与通用的 8 位或 16 位微处理器直接接口。
- 低功耗：390mW。
- 存放温度：-65℃~+150℃。

## 2. AD574 内部结构

AD574 的内部结构如图 9-25 所示。由图可知，AD574 由两部分组成，一部分是模拟芯片，另一部分是数字芯片。其中模拟芯片由高性能的 AD565 D/A（12 位）转换器和参考电压组成；AD565 的主要特点是快速转换、单片结构、电流输出，建立时间是 200ns。数字芯片由控制逻辑电路、逐次逼近型寄存器和三态输出缓冲器组成。AD574 的转换原理与 ADC0809 基本是一样的，也是采用逐次逼近式原理工作的。

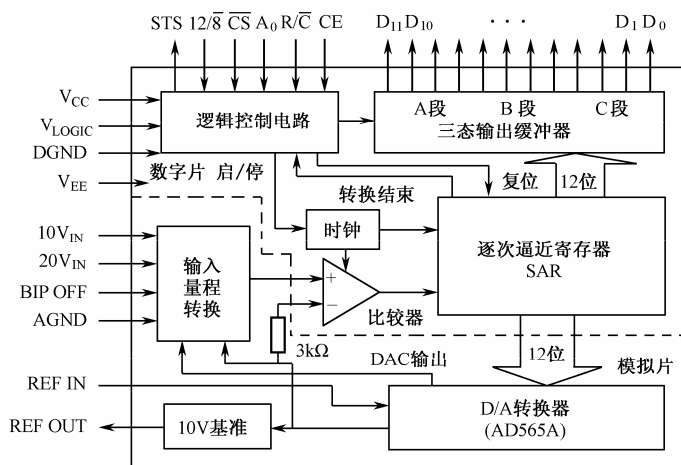


图 9-25 AD574 的内部结构

控制逻辑发出启动/停止时钟信号及复位信号，并控制转换过程。逻辑控制信号受外部 5 个信号及内部转换结束信号的控制。整个转换过程结束后，输出一个标志状态 STS（低电平有效，当 STS=0 时表示 A/D 转换结束）。

当转换开始信号（START）变为高电平时，最高位  $D_{11}$  位开始置“1”，而  $D_{10} \sim D_0$  位全部为“0”状态。经过两个时钟周期， $D_{11}$  下降沿开始对最高位进行判断，也就是决定该位为“1”，还是为“0”。并同时开始对  $D_{10}$  位置“1”，过一个周期后又对该位进行判断，如此进行下去，直到最低位  $D_0$  位为止。

另外，当 START 信号有效时，标志状态 STS 在自动启动后为高电平，当转换结束时变为低电平。

## 3. AD574 引脚功能

AD574 引脚排列如图 9-26 所示。各引脚功能如下：

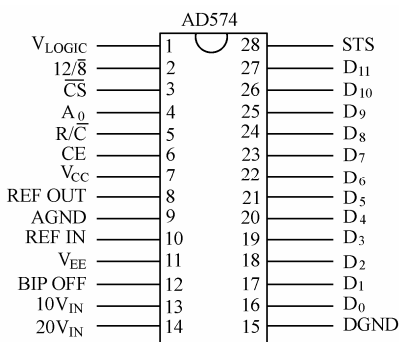


图 9-26 AD574 引脚排列

## (1) 工作电源及“地”

$V_{CC}$ : 正工作电源, +12V/+15V。

$V_{EE}$ : 负工作电源, -12V/-15V。

AGND: 模拟地。

DGND: 数字地。

(2) 逻辑电平端  $V_{LOGIC}$ 

$V_{LOGIC}$  为 +5V 电源。虽然 AD574 使用的工作电源为  $\pm 12V$  或  $\pm 15V$ , 但数字量输出及一些控制信号的逻辑电平仍可直接与 TTL 兼容。需要注意的是, 此 +5V 电源的“地”要和数字量的“地” DGND 连在一起。

## (3) 参考电压输出端 REF OUT 和参考电压输入端 REF IN

AD574 具有内部参考电压, 内部参考电压为  $10 \pm 1\%V$ 。使用时, 由参考电压输出端通过电阻接至参考电压输入端。此参考电压还可供外部电路使用。工作电压为  $\pm 15V$  时, 最大可供电流为 1.5mA, 如果工作电压为  $\pm 12V$ , 则需经过缓冲器。

(4) 模拟量输入端  $10V_{IN}$ 、 $20V_{IN}$  和极性选择端 BIP OFF

当模拟量输入为单极性  $0 \sim 10V$  或双极性  $-5V \sim +5V$  时, 使用  $10V_{IN}$  端; 当模拟量输入为单极性  $0 \sim 20V$  或双极性  $-10V \sim +10V$  时, 使用  $20V_{IN}$  端。

如果模拟量输入为单极性, 则极性选择端 BIP OFF 接 AGND (模拟量的“地”) 即可。若为双极性, 则 BIP OFF 需接  $+5V \sim +10V$  电压。

(5) 数字量输出端  $D_{11} \sim D_0$ 

在 AD574 内部结构中, 三态输出缓冲器分为三段: A 段为高 4 位 ( $D_{11} \sim D_8$ ), B 段为中 4 位 ( $D_7 \sim D_4$ ), C 段为低 4 位 ( $D_3 \sim D_0$ )。

(6) 控制信号端 CE、 $\overline{CS}$ 、 $R/\overline{C}$ 

CE 为芯片使能端, 高电平有效。 $\overline{CS}$  为片选端, 低电平有效。只有当  $CE=1$  且  $\overline{CS}=0$  时, AD574 才能工作。 $R/\overline{C}$  是读出/转换信号。当 CE 和  $\overline{CS}$  同时有效时,  $R/\overline{C}=0$ , 为 AD574 转换操作, 即控制 AD574 开始转换; 而  $R/\overline{C}=1$ , 为读出操作, 控制 AD574 送出转换结果。

(7) 寄存器控制端  $12/\overline{8}$ 、 $A_0$ 

$12/\overline{8}$  为数据输出格式选择信号。当  $12/\overline{8}=1$  时, 若  $R/\overline{C}=1$  (即进行读出操作), 则 AD574 一次送出 12 位的数据转换结果; 当  $12/\overline{8}=0$  时, 若  $R/\overline{C}=1$ , 则 AD574 一次送出 8 位数据转换结果。需要指出,  $12/\overline{8}$  引脚信号与 TTL 不兼容, 不能由 TTL 电平来控制, 必须直接接至 +5V 或数字地 DGND。另外, 此引脚的状态只对读出操作起作用, 对转换操作并不起作用。

$A_0$  引脚的状态有如下两个作用:

第一, 在转换操作开始时,  $A_0$  的状态用于控制转换字长。此时, 若  $A_0=1$ , 则 AD574 进行 8 位转换, 所需时间大约为  $16\mu s$ ; 若  $A_0=0$ , 则 AD574 进行 12 位转换, 所需时间大约为  $25\mu s$ 。这种对转换字长的控制与  $12/\overline{8}$  的状态无关。在转换操作开始之前, 应按要求设定好  $A_0$  的状态, 并使  $A_0$  的状态保持到输出状态信号 STS 变成高电平为止。

第二，在转换数据读出操作中， $A_0$  的状态决定 8 位数据输出的格式。此时，若  $A_0=0$ ，则高 8 位数据输出，同时屏蔽低 4 位。若  $A_0=1$ ，则低 4 位数据输出，并且中 4 位输出全为“0”，同时高 8 位的数据被屏蔽。需要注意的是，如果  $12/\bar{8}=1$ （12 位数据输出格式），则  $A_0$  的状态不起作用。另外，在读出数据过程中，不应改变  $A_0$  的状态，否则可能损坏 AD574 的输出缓冲器。

#### （8）状态输出端 STS

STS 用于表示 AD574 现行工作状态。STS=1 表示 AD574 正在转换，STS=0 表示转换完成。

在  $R/\bar{C}$  由高电平变为低电平时，即有一个负跳变，启动 AD574 进行 A/D 较换。AD574 控制逻辑接到开始转换信号时，它将使时钟有效；一旦开始转换，经延时 500ns，STS 变为高电平，此时不应从输出缓冲器输出数据。在转换过程中，AD574 不能再被停止或重新启动。当转换完毕后，使 STS 电平变低。此时输出缓冲器中的数据可由外部命令读出。

综上所述，AD574 的启动过程是：当  $\overline{CS}=0$ ， $CE=1$ ， $R/\bar{C}=0$  时启动转换。 $A_0=0$  时，启动 12 位转换； $A_0=1$  时启动 8 位转换。在转换期间 STS 为高电平，转换结束后 STS 为低电平。启动时序如图 9-27 所示。

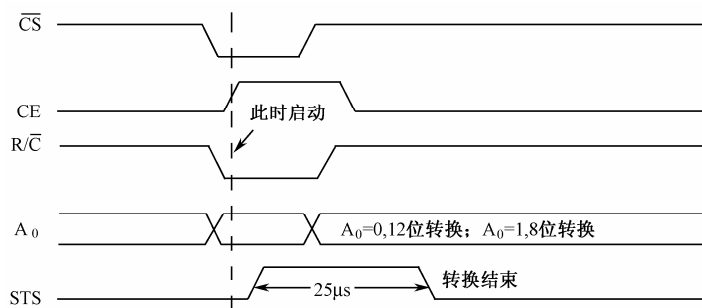


图 9-27 AD574 启动时序

读 AD574 的数据过程是：当  $\overline{CS}=0$ ， $CE=1$ ， $R/\bar{C}=1$  时读取数据。当  $12/\bar{8}=1$  时，一次读出 12 位数据。当  $12/\bar{8}=0$  时，12 位数据分两次读出： $A_0=0$  时读取高 8 位有效位； $A_0=1$  时，读取低 4 位有效位。

#### 4. AD574 工作方式

在转换周期，模拟输入的电流要受 DAC 的约 500kHz 试验电流的调制影响，因此要求模拟输入信号源能在动态负载情况下保持稳定的输出电压。如果信号源达不到此要求，则可以加一级宽频带的运算放大器，也可在普通运算放大器的反馈回路中加入三极管或集成缓冲器。

AD574 有两种工作方式：单极性模拟输入和双极性模拟输入，其电路结构如图 9-28 所示。

##### （1）单极性模拟输入

对于单极性模拟输入，0~10V 量程使用  $10V_{IN}$  引脚，0~20V 量程使用  $20V_{IN}$  引脚。

AD574 单极性输入电路接法如图 9-28(a)所示。

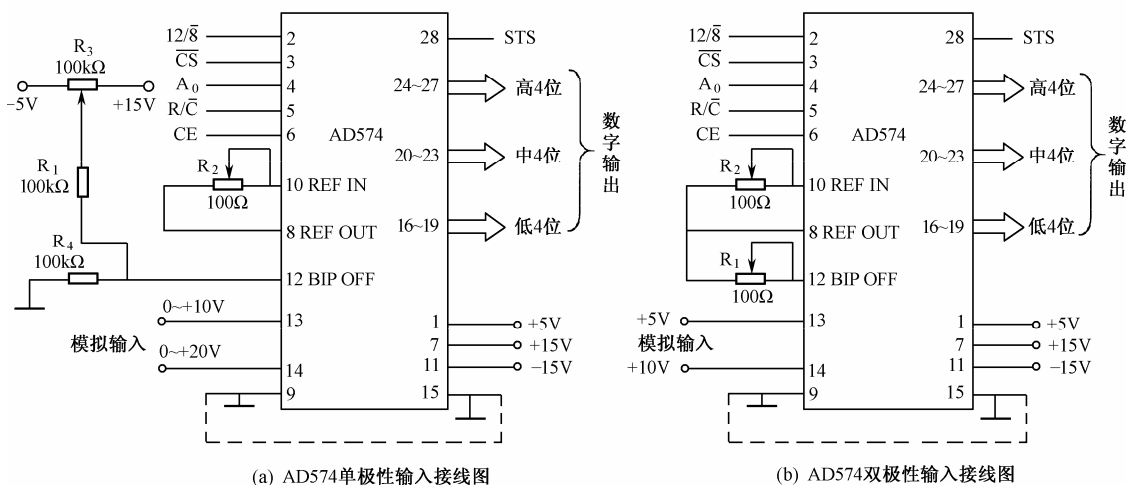


图 9-28 AD574 工作方式

在图 9-24(8)中,若需要进行零调整,则 BIP OFF 通过电阻接工作电源,并由电位器  $R_3$  进行调整。若需要进行满量程调整,则 REF OUT 与 REF IN 之间用电位器  $R_2$  调节。若不需零调整,可将 BIP OFF 接 AGND。若不需满量程调整,可在 REF OUT 和 REF IN 之间接一固定的  $50\Omega$  金属膜电阻。

## (2) 双极性模拟输入

对于双极性模拟输入,  $-5V \sim +5V$  量程使用  $10V_{IN}$  引脚,  $-10V \sim +10V$  量程使用  $20V_{IN}$ 。AD754 双极性输入电路接法如图 9-28(b)所示。

在图 9-28(b)中,若需要进行零调整,则 BIP OFF 与 REF OUT 之间接一电位器  $R_1$  ( $100\Omega$ ) 进行调整。若需要进行满量程调整,则 REF OUT 与 REF IN 之间接一电位器  $R_2$  ( $100\Omega$ ) 进行调整。若不需零调整,或不需要满量程调整,则把其中的电位器  $R_1$  和  $R_2$  换成 2 个  $50\Omega$  金属膜电阻即可。

## 5. AD574 接口设计

图 9-29 是 AD574 与 8088CPU 的单极性接口电路图,模拟量信号可为  $10V$  或  $20V$ 。8088 CPU 通过常规的总线驱动得到三总线信号,通过 74LS138 译码器进行端口地址的寻址,各端口的地址如表 9-6 所示。 $12/\overline{8}$  引脚接地,表示该芯片在数据输出时,每次输出 8 位数据。AD574 的  $R/\overline{C}$  和  $A_0$  分别接到 8088 CPU 的  $A_1$  和  $A_0$  上,从而用不同的端口地址来实现对 AD574 的启动转换和数据输出。AD574 的工作状态由 STS 引脚输出,可将 STS 经三态门电路(比如 74LS244)连至数据总线,比如连到  $D_0$  上,于是可以用查询的方法判断 AD574 是否转换完成,可否读入数据。

综上所述,若要求对 AD574 进行 12 位 A/D 转换,连续采样 100 次,转换结果依次存入  $2000H:1000H$  开始的内存单元,可写出 AD574 在查询方式下的转换程序:

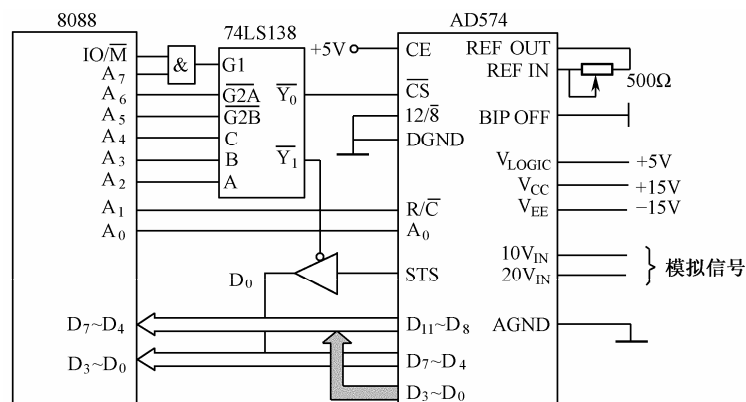


图 9-29 AD574 与 8088CPU 单极性接口电路

表 9-6 端口地址

A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub> (R/C-bar)	A <sub>0</sub> (A <sub>0</sub> )	功能说明	端口地址
1	0	0	0	0	0	0	0	启动 A/D 12 位转换器	80H
						0	1	启动 A/D 8 位转换器	81H
						1	0	读高 8 位转换结果	82H
						1	1	读低 4 位转换结果	83H
1	0	0	0	0	1	×	×	读状态	84H~87H

```
STA: MOV     AX, 2000H
      MOV     DS, AX
      MOV     DI, 1000H
      MOV     CX, 100
X1:   OUT     80H, AL
      NOP
X2:   IN      AL, 84H
      TEST    AL, 01H
      JNZ     X2
      IN      AL, 82H
      MOV     AH, AL
      IN      AL, 83H
      MOV     [DI], AX
      ADD     DI, 2
      LOOP    X1
      HLT
```

9.5.4 双积分式A/D转换器

在各类数字仪表及低速采集系统中，集成双积分式 A/D 转换器由于其优越的性能而被

广泛使用。MC14433(5G14433)采用双积分原理完成 A/D 转换,全部转换电路用 CMOS 大规模集成电路技术设计,具有功耗低、精度高、功能完整、使用简便,以及与微机或其他数字电路兼容等优点。

### 1. MC14433 主要特性

- 转换精度: 读数的  $\pm 0.05\% \pm 1$  字 ( $3\frac{1}{2}$  位十进制数相当于 11 位二进制数)。
- 电压量程: 1.999V 和 199.9mV 两挡。量程扩展通过外加控制电路实现。
- 转换速率: 4~10Hz, 相应时钟频率为 50~150kHz。
- 输入阻抗: 大于 100M $\Omega$ 。
- 工作电压:  $\pm 4.5 \sim \pm 8$  V 或 9~16V。
- 典型功耗: 当电压为  $\pm 5$  V 时, 功耗为 8mW。
- 外型封装: 24 引脚双列直插式。
- 外接单正电源, 基准电压值和量程有关。当量程为 1.999V 时, 基准电压为 2V; 当量程为 199.9mV 时, 基准电压为 200mV。

• 转换结果输出形式为经过多路调制的 BCD 码, 并有多路调制选通脉冲输出, 通过外接译码电路可实现 LED 动态扫描显示或 LCD 显示。

### 2. 数字输出方式

MC14433 的引脚逻辑如图 9-30 所示。

MC14433 转换的结果采用 BCD 码动态扫描输出, 它的千位、百位、十位、个位 4 个数位分别与 DS<sub>1</sub>、DS<sub>2</sub>、DS<sub>3</sub>、DS<sub>4</sub> 输出高电平时相对应。电路中 DS<sub>1</sub> 与 DS<sub>2</sub>、DS<sub>2</sub> 与 DS<sub>3</sub>、DS<sub>3</sub> 与 DS<sub>4</sub> 之间分别相隔两个时钟周期, 每个选通脉冲 (DS<sub>1</sub>~DS<sub>2</sub>) 占 18 个时钟周期。MC14433 输出数据除了在选通信号 DS<sub>2</sub>、DS<sub>3</sub>、DS<sub>4</sub> 对应位上, Q<sub>3</sub>、Q<sub>2</sub>、Q<sub>1</sub>、Q<sub>0</sub> 输出 BCD 码之外, 还在选通信号 DS<sub>1</sub> 对应位上输出“千”位数和别的信号 (欠、过量程标志, 正、负极性标志等)。

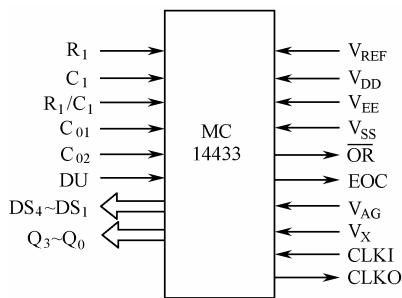


图 9-30 MC14433 的引脚逻辑

## 9.6 闭环控制系统

闭环控制系统一般情况下由控制对象、输入通道、输出通道和计算机系统构成。输入通道的作用是对信息的采集, 输出通道对控制对象加以控制。计算机系统主要用来对输入通道输入的信息进行加工和处理, 然后通过输出通道对控制对象进行控制。

例如, 一个简单的温度控制系统就是一个闭环控制系统。该控制系统由控制对象、测温传感器、加热装置、加热装置的控制部件、制冷装置、制冷装置的控制部件、D/A 转换器(1)、D/A 转换器(2)、A/D 转换器、计算机系统、显示、打印机等组成。温度控制系统原理框图如 9-31 所示。

该温度控制系统工作过程是: 测温传感器把控制对象的温度模拟量 (非电量) 转变成电模拟量, 送给 A/D 转换器转变成数字量; 计算机系统接收到该数字量后经过加工和处理



(处理的算法有多种, 如 PID 算法、数字滤波等), 输出数字量给 D/A 转换器(1)或 D/A 转换器(2), 驱动加热装置的控制部件或制冷装置的控制部件对控制对象加热或制冷。具体控制如下: 当控制对象的温度“高于”设定温度时, 计算机系统经输入通道采集的数据大于设定值, 计算机系统向 D/A 转换器输入数据, 经 D/A 转换器转换成模拟量, 驱动制冷装置的控制部件、控制制冷装置, 使控制对象降温。反之, 使控制对象升温。

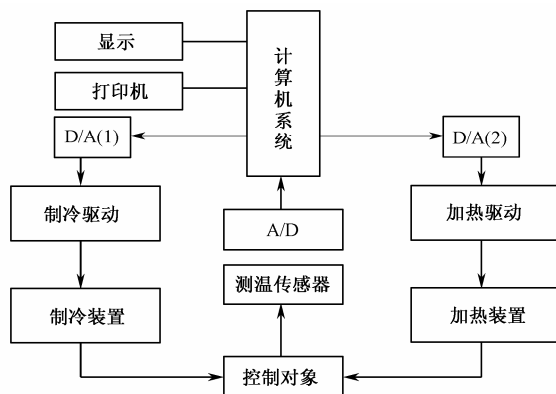


图 9-31 温度控制系统原理框图

## 习题

1. 8255A 的 3 个数据端口在使用时有什么差别?
2. 8255A 有哪几种基本工作方式? 对这些工作方式有些什么规定?
3. 8255A 的工作方式 2 用在什么场合? 说明端口 A 工作在方式 2 时各信号之间的时序关系。
4. 设 8255A 4 个端口地址分别为 110H、112H、114H、116H。要求设置端口 A 为方式 1 的输入, 端口 B 为方式 0 的输出, 端口 C 的高 4 位配合端口 A 工作, 端口 C 的低 4 位为输入。请写出初始化程序。
5. 什么叫异步通信方式? 什么叫同步通信方式? 它们的数据通信格式各有什么特点?
6. 设计一个采用异步通信方式输出 100 个字符的程序段。规定 7 个数据位, 1 个停止位, 用偶校验, 波特率为 9600。8250A 的端口地址为 40H~48H, 输出字符缓冲区地址为 2000H:3000H。
7. 8155 的可编程 I/O 口有哪几种基本的工作方式? 8155 可编程定时器有哪几种工作方式?
8. A/D 转换器和 D/A 转换器在微型计算机应用系统中起什么作用?
9. 采用 DAC0832 电路, 设计并编程完成一个周期可调的梯形波发生器的功能。
10. A/D 转换中, 比较计数法、逐次逼近法的优缺点各是什么?
11. 如果 ADC0809 与微机接口采用查询方式, EOC 应如何与微处理器连接? 转换程序又应如何设计?

# 第 10 章

## 外部设备及其接口

### 📖 教学目的和要求

本章主要以键盘、LED 显示、打印机、视频系统、鼠标、扫描仪、绘图仪等外部设备为典型例子，介绍各种外部设备的结构、性能、功能、工作原理，以及各种外部设备与计算机之间的简单接口。要求了解外部设备的基本性能和应用特点。

## 10.1 概述

在计算机系统中，除中央处理机之外的设备都叫外部设备。外部设备承担着计算机外部各种媒体形式的信息与计算机内部数字信号之间转换的重任。外部设备也是人与计算机沟通与联系的接口，是计算机发挥作用的通道与桥梁。计算机主机的发展离不开外部设备的进步。现代微机系统中，外部设备所占的比重越来越大，其价值占微机系统成本的50%~85%。随着计算机技术的不断发展，许多高新技术在外部设备中得到应用，外部设备产品的范围也越来越广。

计算机外部设备多种多样，型号不同、规格不同、性能不同、价格不同，同一种设备在结构上、工作原理上、操作方法上相差很大，而且在高性能的外部设备上往往采用了最新的科技成果，致使其涉及光、机、电、信息等多种技术领域，技术含量越来越大。因此，学习一些有关外部设备工作原理、基本结构、使用方法等基础知识，对于计算机的应用与开发、外部设备的设计与维护是非常重要的。

## 10.2 键盘及其接口

键盘是微机系统上最基本的标准输入设备。用户通过键盘向计算机输入操作命令、程序或数据。尽管目前已有语音输入、手写板输入、图像扫描识别等多媒体输入方式，然而键盘的重要地位还不会被其他输入方式所取代。

键盘由一组排列整齐的按键（开关）组成，这些按键有数字键（0~9）、字母键（A~Z）、运算键，以及若干个控制键、功能键。按编码提供方式，常用的键盘有两种基本类型，即编码键盘和非编码键盘。

编码键盘能够由硬件逻辑自动提供与被按键对应的 ASCII 码或其他编码。编码键盘中的某一按键被按下后，能够提供与该按键相对应的编码信息。如果是 ASCII 码键盘，就能提供与该按键相对应的 ASCII 码。编码键盘的缺点是硬件设备随着键数的增加而增加。

非编码键盘仅能简单地提供被按键行和列的矩阵，其他工作都靠程序实现。这样，非编码键盘就为系统软件在定义键盘的某些操作上提供了更大的灵活性。目前已有一些专用芯片可以完成其中的一些工作。非编码键盘具有价格便宜、配置灵活的特点。

本节着重介绍非编码键盘的工作原理及接口的结构。

在非编码键盘中，为了检测哪个键被按下，必须解决如下问题：

- （1）清除键接触时产生的抖动干扰。
- （2）防止键盘操作的重键错误。
- （3）键盘的结构及被按键的识别。
- （4）产生被按键相应的编码。

### 10.2.1 消除抖动及重键处理

键盘的按键有机械式、电容式、薄膜式等多种，但就它们的作用而言，都是一个使电

路“通”或“断”的开关。在对机械式按键进行键盘输入时，一般存在两个问题，即触点弹跳与同时按下下一个以上键的问题，也就是所谓的抖动与重键的问题。

### 1. 抖动

抖动是开关本身的一个最普遍的问题，它产生的原因是，当机械开关的触点闭合时，在达到稳定之前需要短暂抖动或弹跳几下，即反复闭合、断开几次之后，才能可靠地闭合在一起。抖动也存在于开关断开时，其情形与开关闭合时相同。

抖动产生的尖脉冲情况如图 10-1 所示。

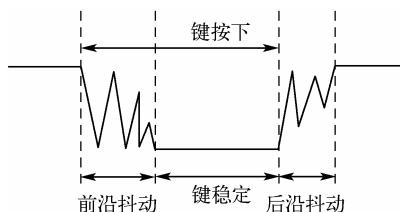


图 10-1 按键抖动波形

根据所用按键的不同质量，按键的抖动时间可为 10~20ms。按键的抖动会引起一次按键被读入多次。解决按键的抖动可以使用硬件滤波方法或软件延迟方法。硬件滤波是对每一个按键加上 RC 滤波电路，或加上 RS 去抖电路。这种方法通常在键数少的情况下使用。而键数较多时，则经常采用软件去抖动技术，这种方法的实质就是采用一个能产生 20ms 左右延迟的子程序，以等待键的输出达到完全稳定后才去读取代码。

### 2. 重键

所谓重键是指两个或两个以上的键同时按下，或者一个键按下后还未弹开，另一个键又按下的情况。

由于操作上的原因，在键盘上同时按下下一个以上的键是可能的（组合键除外）。检测出这种现象并防止产生错误编码是很重要的。解决这个问题的三种主要技术是：两键同时按下保护技术、 $n$  键同时按下保护技术和  $n$  键连锁技术。

第一种保护技术为同时按下两键的场合提供保护。最简单的处理方法是，当只有一个键按下时才读取键盘的输出，并且认为最后仍被按下的键是有效的正确按键。这种方法常用于软件扫描键盘场合。当采用硬件技术时，往往采用锁定的方法。锁定保护方法的原理是，当第一个按键未松开时，按第二个按键不起作用，不产生选通脉冲。这可以通过锁定内部延迟机构来实现，锁定的时间和第一个按键的闭合时间相同。

$n$  个按键同时按下的保护技术有两种处理方法，一种是不理会所有被按下的按键，直至只剩下一个按键按下时为止；另一种是将所有按键的信息存入内部键盘输入缓冲器，逐个处理，这种方法成本较高，在较便宜的系统中很少采用。

对于  $n$  键连锁（锁定）技术，当某一个按键被按下时，在此按键未完全释放之前，其他的按键虽然可被按下或松开，但并不产生任何代码，这种方法实现起来比较简单，因而比较常用。

### 10.2.2 线性键盘

根据按键的连接方式，键盘可以分为线性键盘和矩阵键盘两类。

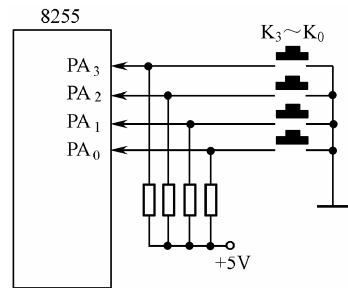


图 10-2 线性键盘按键电路

线性键盘采用独立式按键，是最简单的键盘结构，它是指直接用 I/O 口线构成的单个按键电路。每一键互相独立地各自接通一条输入 I/O 口线，每条 I/O 口线上的按键的工作状态不会影响其他 I/O 口线的工作状态。如图 10-2 所示为线性键盘的按键电路。通常按键输入都采用低电平有效，上拉电阻保证了按键断开时，I/O 口线有确定的高电平。当 I/O 口内部有上拉电阻时，外电路可以不配置上拉电阻。

线性键盘电路配置灵活，软件结构简单。但每个按键必须占用一根 I/O 口线，在按键数量较多时，I/O 口线浪费较大。故在按键数量不多时，常采用这种按键电路。

以图 10-2 的电路连接为例，假设 8255 的 A 口、B 口、C 口、控制口的端口地址分别是 60H、61H、62H、63H，采用软件消抖技术（只考虑前沿消抖），编程实现对按键 K<sub>3</sub>~K<sub>0</sub> 的识别。假设按键 K<sub>3</sub>~K<sub>0</sub> 的对应编码为 3~0，识别按键后，将对应的编码存到 AH 寄存器中。有 D20ms 延时子程序可以调用。程序如下。

```
CODE    SEGMENT
        ASSUME  CS:CODE
KEY      PROC    FAR
START:   PUSH    DS
        MOV     AX, 0
        PUSH    AX
        MOV     AL, 90H
        OUT     63H, AL        ; 设置 8255 的 A 口为方式 0，输入
X1:      IN      AL, 60H        ; 输入 A 口键盘状态
        AND     AL, 0FH        ; 析取 K3~K0 信号线
        CMP     AL, 0FH
        JZ      X1            ; 没有键按下，继续查询
        CALL    D20ms         ; 有键按下，延时消抖
        IN      AL, 60H        ; 输入 A 口键盘状态
        AND     AL, 0FH        ; 析取 K3~K0 信号线
        CMP     AL, 0FH
        JZ      X1            ; 此时，说明延时消抖前的按键判断是源于干扰，
                                ; 或者，延时消抖时间不足，重新查询
        CMP     AL, 00001110B
        JNZ     X2            ; 不是单键 K0 按下，转
        MOV     AH, 0         ; 设置 K0 的编码
```

```

                JMP      XEND
X2:             CMP      AL, 00001101B
                JNZ      X3          ; 不是单键 K1 按下, 转
                MOV      AH, 1
                JMP      XEND
X3:             CMP      AL, 00001011B
                JNZ      X4          ; 不是单键 K2 按下, 转
                MOV      AH, 2
                JMP      XEND
X4:             CMP      AL, 00000111B
                JNZ      X5          ; 不是单键 K3 按下, 转
                MOV      AH, 3
                JMP      XEND
X5:             MOV      AH, 0FFH    ; 此时说明有多个按键同时按下,
                                     ; 用 0FFH 表示这种状态
XEND:           NOP                  ; 在此处可加入其他需要处理的程序
                RET
KEY             ENDP
CODE            ENDS
END             START

```

### 10.2.3 矩阵键盘

为了减少键盘接口所占用 I/O 线的数目, 在按键数较多时, 通常都将按键排列成矩阵形式。矩阵键盘又叫行列式键盘, 用 I/O 口线组成行列结构。按键设置在行列的交点上。例如,  $2 \times 2$  的行列结构可构成 4 个键的键盘,  $4 \times 4$  的行列结构可构成 16 个键的键盘。利用这种矩阵结构只需  $N + M$  条 I/O 口线, 即可连接  $N \times M$  个按键。

在这种矩阵键盘结构中, 对按键的识别是对键盘扫描后, 通过软件来完成的。键盘扫描方式一般有两种, 一种是传统的行扫描法, 另一种是速度较快的线反转法。

#### 1. 行扫描法

行扫描法是步进扫描方式, 每次向键盘的某一行发出扫描信号, 同时通过检查列线的输出来确定闭合键的位置。图 10-3 给出了一个  $4 \times 4$  的键盘矩阵, 共有 16 个键。假设其中第 2 行第 2 列的键 N (2,2) 闭合, 其余断开, 则行扫描的过程是这样的: 微处理器先输出 0000 到键盘的 4 根行线, 由于 N 键闭合, 因而由键盘列线输入到微处理器的代码是 1011, 此时微处理器得知有按键闭合, 且闭合键在第 2 列上, 但不知道在第 2 列的哪一行。为此需进行逐行扫描寻找, 微处理器先发出 1110 以扫描 0 行, 此时输入为 1111, 表示被按的键不在第 0 行; 第二次输出 1101, 扫描第 1 行, 输入仍为 1111, 表示被按的键不在第 1 行; 第三次输出 1011, 输入为 1011, 表示被按的键在第 2 行第 2 列上。这样微处理器得到一组输出/输入代码——1011(出)/1011(入), 它可以确定按键所在的位置, 因而称之为按键的位

置码（键值）。通常位置码不同于按键的读数（键号），因而必须用软件进行转换，这可借助于查表或其他方法完成。

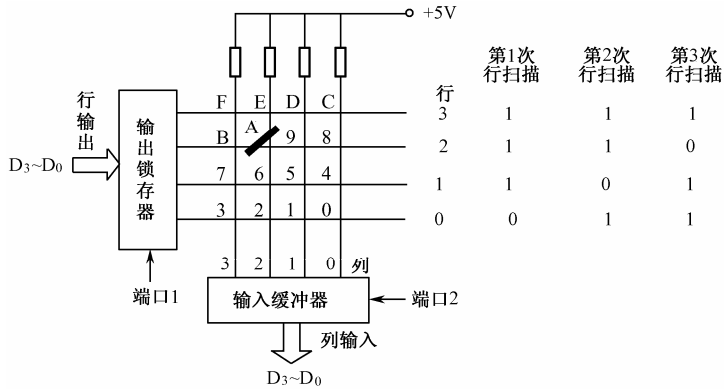


图 10-3 行扫描法原理图

以图 10-3 的电路连接为例，假设行输出端口 1 的地址为 200H，列输入端口 2 的地址为 201H，采用软件消抖技术（只考虑前沿消抖），编程实现对 0~F 键的识别。识别按键后，将按键的键号（即 0~F）存到 AH 寄存器中；若为重键，则将 0FFH 存到 AH 寄存器中。有 D20ms 延时子程序可以调用。

本例中，键的位置码是由行号和列号组合而成的 1 字节的数据，4 位行号占据键位置码的高 4 位，4 位列号占据键位置码的低 4 位。例如，B 键的行号为 1011，列号为 0111，则 B 键的位置码为 10110111。

程序如下。

DATA	SEGMENT	
TABLE	DB	11101110B ; 第 0 行第 0 列, 0 键的位置码
	DB	11101101B ; 第 0 行第 1 列, 1 键的位置码
	DB	11101011B ; 第 0 行第 2 列, 2 键的位置码
	DB	11100111B ; 第 0 行第 3 列, 3 键的位置码
	DB	11011110B ; 第 1 行第 0 列, 4 键的位置码
	DB	11011101B ; 第 1 行第 1 列, 5 键的位置码
	DB	11011011B ; 第 1 行第 2 列, 6 键的位置码
	DB	11010111B ; 第 1 行第 3 列, 7 键的位置码
	DB	10111110B ; 第 2 行第 0 列, 8 键的位置码
	DB	10111101B ; 第 2 行第 1 列, 9 键的位置码
	DB	10111011B ; 第 2 行第 2 列, A 键的位置码
	DB	10110111B ; 第 2 行第 3 列, B 键的位置码
	DB	01111110B ; 第 3 行第 0 列, C 键的位置码
	DB	01111101B ; 第 3 行第 1 列, D 键的位置码
	DB	01111011B ; 第 3 行第 2 列, E 键的位置码
	DB	01110111B ; 第 3 行第 3 列, F 键的位置码

; 了干扰), 以 80H 作为这种情况的标志。



；该指令的设置，主要考虑到程序的完备性，  
；即可以使程序在任何情况下都能正确执行。

```

      JMP      XEND
X3:   MOV      CL, 4
      SHL      AH, CL      ; AH 逻辑左移 4 位，将低 4 位的行号移到高 4 位
      OR       AL, AH      ; 行号与列号相“或”，形成键的位置码
      LEA      BX, TABLE  ; 设置 TABLE 位置码表的指针
      MOV      CL, 0      ; 设置键号初值为 0
X4:   CMP      AL, [BX]    ; 在 TABLE 表中查找本次形成的键位置码
      JZ       X5          ; 找到，转，对应的键号就在 CL 中
      INC      CL          ; 未找到，键号加 1
      INC      BX          ; 指向下一个存储单元保存的键位置码
      CMP      CL, 10H     ; 键号等于 10H 吗？
      JNZ      X4          ; 不等，继续查找
      MOV      AH, 0FFH   ; CL 等于 10H，说明在 TABLE 表中没有找到
                                ; 对应的键位置码，其原因可能是出现了重键
                                ; 的情况，以 0FFH 作为这种情况的标志。
      JMP      XEND
X5:   MOV      AH, CL      ; 将 CL 中保存的键号传到 AH 中
XEND: NOP
      RET
KEY   ENDP
CODE  ENDS
      END      START
```

计算机系统工作时，并不经常需要键输入，因此，在查询方式的行扫描法中，CPU 经常处于空扫描状态。为了进一步提高 CPU 效率，可以采用如图 10-4 所示的中断行扫描法工作方式。

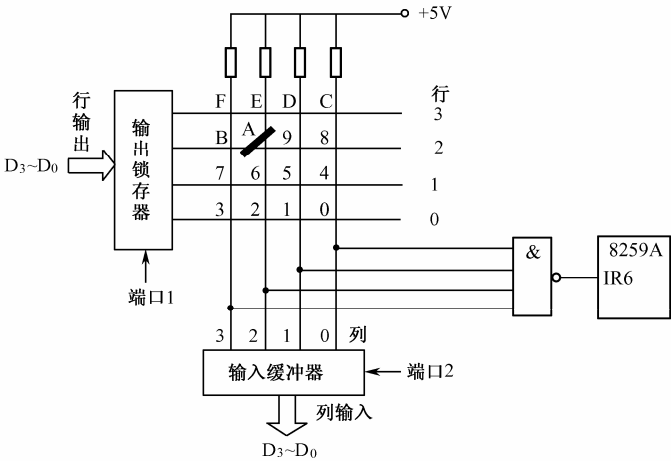


图 10-4 行扫描法

该电路的工作原理：列线通过上拉电阻接+5V 时，被钳位在高电平状态。行输出锁存器将所有行线置成低电平。列线电平状态经“与非门”送入 8259A 的中断申请端 IR6。如果没有任何键按下，所有列线电平均为高电平，则“与非门”输出为低电平。如果有键按下，总会有一根列线为低电平，“与非门”输出由低电平变高电平，即发出中断请求。若 CPU 开放外部中断，则响应中断请求，进入中断服务程序。在中断服务程序中除完成键识别、键功能处理外，还须有消除键抖动影响、重键处理等措施。

## 2. 线反转法

行扫描法要逐行扫描查询，当所按下的键在最后一行时，则要经过多次扫描才能获得键值。而采用线反转法时，只要经过两个步骤即可获得键值。这种方法需要利用一个可编程的输入/输出接口，例如 Intel 8255A 芯片等。线反转法的基本原理如图 10-5 所示。

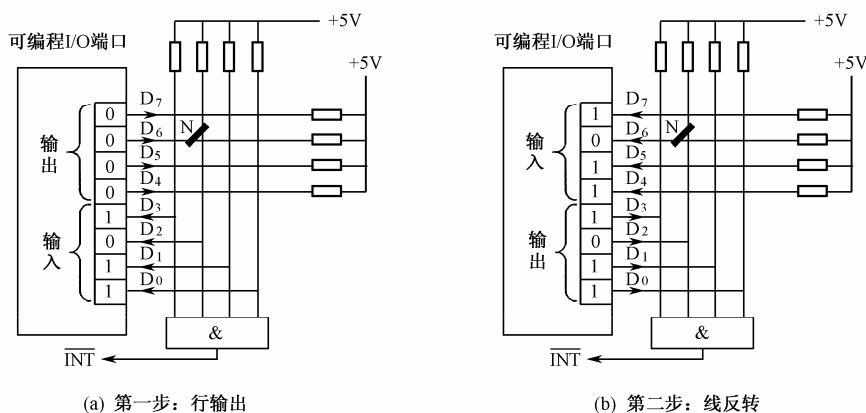


图 10-5 线反转法原理图

整个键的识别过程分两步进行：

第一步，输出行信号。首先将 I/O 口编程，指定  $D_3 \sim D_0$  为列输入线， $D_7 \sim D_4$  为行输出线，并使行输出信号  $D_7 \sim D_4$  为 0000。若按键 N 被按下，这时，与门的一个输入端变为低电平，结果使  $INT=0$ ，向 CPU 请求中断，表示键盘中已有按键被按下。与此同时， $D_3 \sim D_0$  的锁存器锁存了列的代码 1011。显然，其中的“0”对应着被按键 N 的列位置。但仅有列位置还不能识别被按键的确切位置，还必须进一步找出它的行位置。

第二步，线反转。为确定按键的行位置，可以将列代码进行反转传送，通过编程使 I/O 口的输入和输出线完全反转过来，即  $D_3 \sim D_0$  为输出线，而  $D_7 \sim D_4$  为输入线。由图 10-5 可以看到，此时列代码的锁存器将通过  $D_3 \sim D_0$  输出列代码 1011。反转传送的结果使  $D_7 \sim D_4$  得到的输入为 1011，并锁存到相应的寄存器中。 $D_6$  的“0”就指明了被按键 N 的行位置。

至此，I/O 口 8 位数据寄存器的  $D_7 \sim D_0$  为 10111011，既包含了被按键的“行”位置，也包含了被按键的“列”位置，这样被按键 N 就被完全识别出来了。

线反转法的优点是只需要一个非常简单的程序，并且不需要逐行扫描，因而速度比较快。缺点是需要一个专用的可编程 I/O 口作为键盘管理。

### 10.2.4 键盘工作方式

在微机应用系统中，键盘扫描只是 CPU 工作的内容之一。那么 CPU 在忙于各项工作任务时，如何兼顾键盘扫描，以保证既能及时响应键盘操作，又不过多地占用 CPU 的时间呢？这就要根据应用系统中 CPU 的忙、闲情况，选择好键盘的工作方式。键盘的工作方式有三种：

（1）程序控制扫描方式。这种方式是利用 CPU 工作的空余时间，调用键盘扫描子程序，响应键盘的输入请求。

（2）定时扫描方式。这种方式是利用定时器产生定时中断（例如 10ms），CPU 响应中断后对键盘进行扫描，并在有键按下时转入键功能处理程序。定时扫描方式在本质上是中断方式，但不是实时响应，而是定时响应。

（3）中断扫描方式。当应用系统工作时，并不经常需要键的输入，因此，无论键盘是工作于程控方式还是定时方式，CPU 都经常处于空扫描状态。为了进一步提高 CPU 的效率，可以采用中断扫描方式，当键盘上有按键闭合时便产生中断请求，CPU 响应中断，执行中断服务程序，对闭合键进行识别，并做相应的处理。

### 10.2.5 PC 键盘与接口

PC 系列机都采用非编码键盘，其按键排列为矩阵式。不同时期的 PC 系列机配有物理上各不相同的键盘。早期的 PC 和 PC/XT 使用的是具有 83 个按键的键盘，这种键盘一般称做标准键盘。对 80286 以上的机型，一般使用具有 101 个按键的增强型扩展键盘。键盘与微机的接口采用如图 10-6 所示的电缆插头。早期的 PC、PC/XT 和一些增强型扩展键盘使用的是 5 针电缆插头，不过目前大部分都是使用 6 针微型电缆插头，也有些键盘使用 USB 接口。

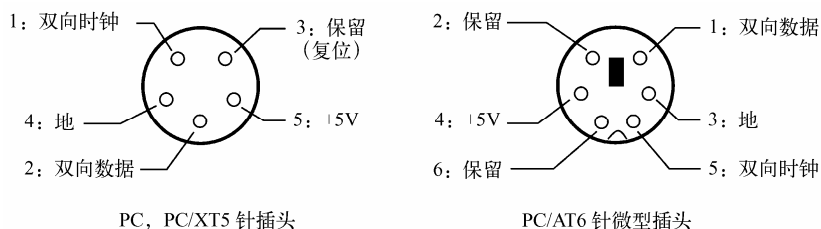


图 10-6 键盘 5 针插头和 6 针微型插头接线

在标准键盘中用 Intel 8048 单片机作为键盘控制器，来控制对键盘的扫描和向微机发送扫描码（键盘输出的数据信号称为扫描码）。扫描码反映键的位置和键的接通或断开状态。一个键的接通与断开分别输出接通扫描码和断开扫描码。主机的键盘接口电路收到一对接通和断开扫描码并判断无误时，才认为是一次正确的键盘输入；否则，就要重新输入。

标准键盘的扫描码用 1 字节表示。接通扫描码是键号的二进制数，断开扫描码由接通扫描码的最高位置 1 形成。如 F 键的接通扫描码为 21H，而断开扫描码为 21H+80H=A1H。

早期的 PC、PC/XT 与键盘的接口采用移位寄存器（74LS322）来接收键盘发送的串行扫描码，通过并行接口 8255A（I/O 口地址 60H）将装配好的数据送给 CPU，同时向 8259A（IRQ1）发中断请求。

在增强型扩展键盘中用 Intel 8048 或 8049 单片机作为键盘控制器，来控制对键盘的扫描和向微机发送扫描码等工作。它的键接通扫描码和键断开扫描码均与标准键盘有很大差距，例如，F 键的接通扫描码表示为 21H，而断开扫描码用 2 字节，在接通扫描码之前加 1 个高位字节 0FOH，即 0F021H。

为了保持键盘中断程序处理按键状态输入的一致性，增强型扩展键盘的接口除应具备标准键盘接口的所有功能外，还应将键盘扫描码转换成兼容的系统扫描码，并能发送和接收一些键盘命令。为了完成这些工作，增强型扩展键盘的微机接口采用了单片机（如 8042）作为键盘接口控制器，负责键盘接口的全部工作，实现键盘与主机之间的双向数据传输。

图 10-7 给出了增强型扩展键盘接口逻辑示意图。

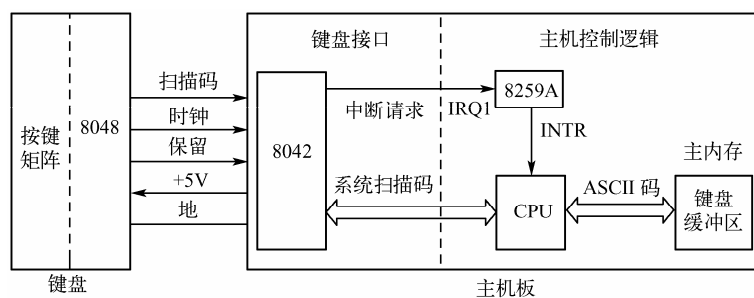


图 10-7 增强型扩展键盘接口逻辑示意图

### 1. 增强型扩展键盘的工作原理

增强型扩展键盘的内部结构电路如图 10-8 所示。由图可以看出，键盘的核心是 1 个 Intel 8048 单片机，作为键盘控制器。8048 是 1 个 8 位的 CPU，主频为 2MHz，内部设有 1KB 的 EPROM、64B 的 RAM、8 级堆栈、1 个可编程的 8 位计数/定时器。外部引脚有 8 条数据线、两个 8 位 I/O 接口 P10~P17 和 P20~P27。

增强型扩展键盘矩阵按 16 行×8 列来排列，用单片机 Intel 8048 来控制对键盘的扫描。工作时，Intel 8048 发出 4 位行计数信号 CNT04、CNT08、CNT16、CNT32 和 3 位列计数信号 CNT04、CNT08、CNT16。行计数信号通过 4-16 行译码器 74LS159 得到 16 个行扫描信号 DR<sub>0</sub>~DR<sub>15</sub>，列计数信号通过 3-8 列译码器 74LS156 得到 8 个列扫描信号 DR<sub>16</sub>~DR<sub>23</sub>。CNT64 作为行 / 列控制信号：低电平时，作为行计数信号，高电平时，则作为列计数信号。扫描输出回送给 8048。8048 对扫描的输入进行判断，无误时，由内部软件查表得到输出扫描码。

键盘与微机键盘接口以串行方式进行通信。传输的数据流由 11 位二进制位串组成。

格式符合异步串行规则，包含 1 位低电平的起始位，8 位数据位，1 位奇校验位，1 位停止位。

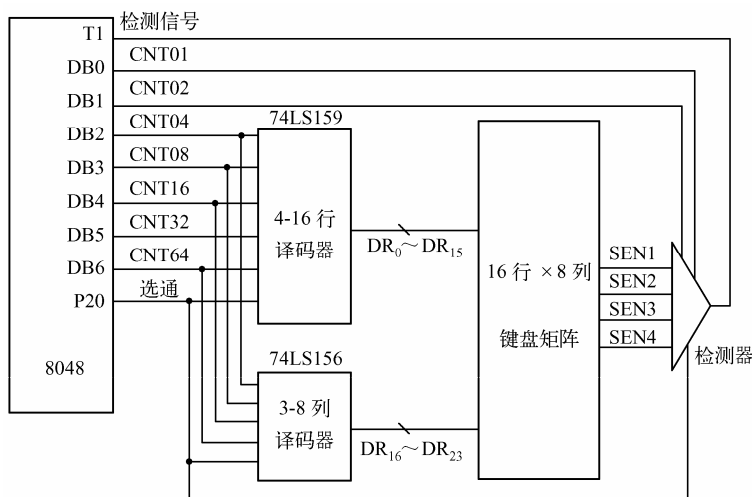


图 10-8 增强型扩展键盘内部结构电路示意图

键盘与微机键盘接口通过时钟线 and 数据线进行双向通信。时钟线的主要作用是传送同步脉冲，传送的数据位串在正脉冲期间有效。数据线用于传送二进制位串数据，在任一时刻，只能有一方发送数据，这一点类似于半双工通信方式。

当有键按下或键盘需要向微机键盘接口回送命令时，键盘进入发送状态。键盘发送数据要处理两种情况：

(1) 发送前，首先检查时钟线和数据线状态。只有时钟线和数据线都是高电平时，才可以正常发送。若时钟线为高电平，数据线为低电平，表明微机键盘接口请求发送，键盘准备接收。

(2) 在发送过程中，键盘同时不断地测试时钟线状态。若时钟线长时间出现低电平状态，键盘立即停止发送。

## 2. 键盘接口逻辑功能

键盘接口逻辑电路的核心是 1 片 8042 单片机。Intel 8042 是一个通用的外围接口处理器，内部包含有 8 位 CPU、2KB 的 ROM、128B 的 RAM、8 位可编程计时器、2 个可编程 8 位 I/O 口 (P10~P17 和 P20~P27) 和 2 个 1 位的输入测试口 TEST0、TEST1。此外，还有 1 个 8 位的状态寄存器和 2 个数据寄存器。8042 可以支持 2 个中断源和 DMA 操作，计时器可用于产生时序信号或对外部信号进行计数。8042 使用的时钟信号是系统时钟经过分频后产生的，频率为 6MHz。

8042 通过内部的状态寄存器、输出缓冲器、输入缓冲器与系统通信。作为系统的一个 I/O 设备，8042 的片选信号  $\overline{CS}$  接至 I/O 口地址译码电路，选用地地址范围为 60H~7FH。在实际应用中，系统使用两个端口地址访问 8042。当 I/O 端口地址为 64H 时，系统可向 8042 输入缓冲器写入命令或从状态寄存器读取状态。当 I/O 地址为 60H 时，系统从输出缓冲器读取来自键盘的扫描码或命令。

8042 单片机作为微机键盘接口电路的控制器，它的作用是接收键盘送出的串行接通、断开扫描码，经检验无误后，将其转换为并行输出的系统扫描码，并将系统扫描码存入它的输出缓冲器，然后发出中断请求，由主机做进一步处理。

8042 的主要功能如下。

(1) 接收键盘送来的扫描码。该扫描码由键盘的双向数据线和双向时钟线送入 8042。输入的扫描码为串行数据，经奇偶校验确认无误后，存入 8042 的 RAM 区。

(2) 将扫描码转换成系统扫描码。按键的接通和断开对应一次按键的全过程，8042 检查连续两个扫描码，确认为同一键的接通和断开扫描码后，调用芯片内 ROM 中的程序将这组扫描码转换成系统扫描码。虽然标准键盘和增强型扩展键盘的扫描码的编码和传送格式不同，但键盘控制器却将其转换成同一种编码，这种供系统使用的编码，称为系统扫描码。系统扫描码为 1 字节，其编码与标准键盘的接通扫描码相同。

(3) 将系统扫描码送输出缓冲寄存器并发出中断请求。系统扫描码形成之后，将其装入 8042 的输出缓冲寄存器，该寄存器与系统的数据总线相连。至此，原先串行输入的数据被转换成并行数据送入输出缓冲寄存器，发出中断请求，让主机取出扫描码并做进一步处理。

(4) 请求重发键盘扫描码。若串行输入的键盘扫描码检验有误，8042 通过双向数据线、双向时钟线以串行方式向键盘发出重发扫描码的命令。在限定的时间（20ms）内键盘不重发扫描码的应答信号，该次输入便作废。

(5) 接收并执行系统命令。可从状态寄存器读出键盘的工作状态，也可从输入缓冲寄存器输入命令或数据，改变键盘的使用方式。

系统通过键盘接口向键盘发送数据时，同样首先要检查时钟信号线。所不同的是如果检测到键盘正在发送数据，接着判断是否已接收到第 10 个二进制位。如果接收到第 10 位，就等待接收完毕。如果接收的位少于 10 个，系统将强制时钟线为低电平，放弃本次接收的数据，准备发送。系统强制时钟线低电平的时间至少要持续 60μs。

## 10.2.6 BIOS 键盘中断及 DOS 键盘功能调用

BIOS 键盘中断及 DOS 键盘功能调用有中断类型码 09H、16H、21H 三种方式。

### 1. 中断 09H 的功能

微机键盘接口电路把来自键盘的串行扫描码变成并行的系统扫描码，送入 8042 的输出缓冲寄存器，并向主 CPU 发出中断请求。当 CPU 响应中断请求后，执行类型码为 09H 的中断服务程序，其功能如下：

(1) 从键盘接口的输出缓冲寄存器（60H）读取系统扫描码。

(2) 判断该键是单独按下或是与组合键（Shift + Ctrl 或 Alt）一起使用。若字符键是单独按下，将扫描码转换为相应的 ASCII 码或扩展码（命令键、组合功能键等的编码，称为扩展码）写入键盘缓冲区。例如，系统扫描码为 1EH，若设有与 Shift 键一起使用，将其转换为 a 的 ASCII 码 61H。若有 Shift 键配合使用，则将其转换为 A 的 ASCII 码 41H。

(3) 如果是换挡键（如 CapsLock, Ins 等），将其状态存入 BIOS 数据区中的键盘标志单元。

(4) 如果是组合键（如 Ctrl + Alt + Del），则直接执行，完成其相应的功能。

(5) 对于中止组合键（如 Ctrl + C 或 Ctrl + Break），强行中止应用程序的执行，返回 DOS。

(6) 将转换的 ASCII 码作为低字节，以原来的系统扫描码作为高字节存入键盘缓冲区，供系统调用。键盘缓冲区建立在系统主存的 BIOS 数据区中，占用 32 字节，可存放 16 次击键产生的 ASCII 码和扫描码。它以先进先出的方式工作，输入的键盘代码在其中形成循环队列。中断 09H 输入的地址指针总指向队尾，从那里写入数据。

(7) 在完成上述任务之后，结束中断调用，中断返回。至此，一次按键输入的信息才真正送入微机之中。

## 2. 中断 16H 的功能

应用程序需要使用存入键盘缓冲区的字符，例如，需要根据输入的字符作为程序的转移条件时，可使用“INT 16H”的软件中断，它以先进先出的方式工作，“INT 16H”的输出指针总指向队列首，从那里取出字符。

“INT 16H”有 3 种子功能，由 AH=0、1、2 识别：

第一，从键盘缓冲区读取 ASCII 码（包括扫描码）。

第二，判断缓冲区是否为空。若缓冲区循环队列的首指针与尾指针相同，意味着缓冲区的键码已经取完，等待输入新的键码。否则，还有未被取走的键码。

第三，判断当前键盘的特殊键（如 Ctrl、Caps Lock）的状态。用户可以使用中断指令“INT 16H”获取相应的键盘状态信息。键盘状态信息由 1 字节表示，如图 10-9 所示。

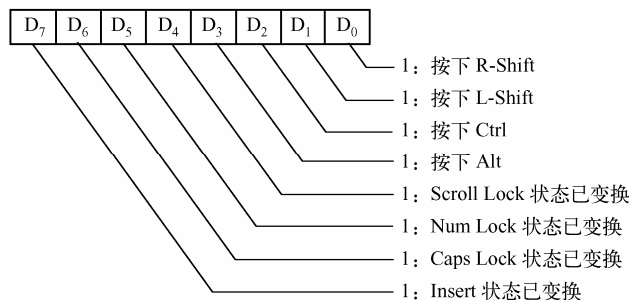


图 10-9 键盘状态字节

### (1) AH=0。

功能：从键盘读入字符送 AL 寄存器，当无键按下时，处于等待状态。

入口参数：AH=0。

出口参数：AL 中为键盘输入的字符的 ASCII 码值，AH 中为扫描码。

### (2) AH=1。

功能：从键盘缓冲器中读入字符送给 AL，并设置 ZF 标志，若按任一键（即键盘缓冲区不空），置 ZF=0，否则 ZF=1。

入口参数：AH=1。

出口参数：若 ZF=0，则 AL 中为输入的字符的 ASCII 码。

由于该功能是从键盘缓冲区读数的，当无键按下时，不等待，常通过检测 ZF 标志来控制某一程序的执行。

### (3) AH=2。

功能：读取特殊功能键的状态。

入口参数: AH=2。

出口参数: AL 为各特殊功能键的状态, 位 7 为 1 是插入键 (INS), 位 6 为 1 是大小写字母键 (Caps), …, 位 1 为 1 是左边的 Shift 键, 位 0 为 1 是右边的 Shift 键。

例如, 检测键盘输入的键是否是回车键的程序段如下:

```
START:  MOV    AH, 0
        INT    16H
        CMP    AL, 0DH
        JZ     X1          ; 是回车, 转 X1
        :
```

### 3. 中断 21H 的功能

在 DOS 功能调用中, 也有多个功能调用号用于获得所需要的键盘信息。常用的键盘操作功能如下:

(1) AH=1。

功能: 从键盘输入一个字符并回显在屏幕上。

入口参数: AH=1。

出口参数: AL=字符。

(2) AH=6。

功能: 读键盘字符 (直接控制 I/O)。

入口参数: AH=6, DL=0FFH (表示输入)。

出口参数: 若有字符可取, AL=字符, ZF=0。若无字符可取, AL=0, ZF=1。

(3) AH=7。

功能: 从键盘输入一个字符, 不回显。

入口参数: AH=7。

出口参数: AL=字符。

(4) AH=8。

功能: 从键盘输入一个字符, 不回显。检测 Ctrl\_Break。

入口参数: AH=8。

出口参数: AL=字符。

(5) AH=0AH。

功能: 输入字符到缓冲区。

入口参数: AH=0AH, DS:DX=缓冲区首址。

出口参数: 无。

(6) AH=0BH。

功能: 读键盘状态。

入口参数: AH=0BH。

出口参数: AL=0FFH, 有键输入。AL=0, 无键输入。



(7) AH=0CH。

功能：清除键盘缓冲区，并调用一种键盘功能。

入口参数：AH=0CH，AL=键盘功能号（1、6、7、8、A）

出口参数：与调用的功能有关。

例如，检测键盘输入的键是否是回车键的程序段如下：

```
START:  MOV    AH, 7
        INT     21H
        CMP     AL, 0DH
        JZ      X1          ; 是回车则转 X1
        :
```

10.3 LED显示器及其接口

10.3.1 七段LED显示器结构

七段 LED 显示器是用发光二极管显示字形的显示器件。在应用系统中通常使用的是七段显示器。七段显示器由七段组成，每一段是一个发光二极管，排成一个“日”字形。通过控制某几个发光二极管的导通发光而显示出某一字形，如数字0~9，字符A、B、C、D、E、F、P等。

通常的七段 LED 显示器有 8 个发光二极管，故也有人称其为八段显示器，如图 10-10 所示。其中 7 个发光二极管构成字形“8”，一个发光二极管构成小数点。

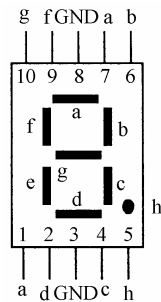


图 10-10 七段 LED 显示器引脚配置

表 10-1 列出了共阴极接法七段显示器的字形和段选码的关系。所谓段选码即控制各段发光二极管的 8 位数据代码，也可称为字形代码。当然，如果要显示其他的符号，也可以排出相应的段选码。另外，如果采用共阳极接法，则段选码是该表数据的取反。

表 10-1 共阴极接法七段显示器的字形和段选码的关系

字 形	h	g	f	e	d	c	b	a	段码 (H)
	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	
0	0	0	1	1	1	1	1	1	3F
1	0	0	0	0	0	1	1	0	06
2	0	1	0	1	1	0	1	1	5B
3	0	1	0	0	1	1	1	1	4F

续表

字 形	h	g	f	e	d	c	b	a	段码 (H)
	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	
4	0	1	1	0	0	1	1	0	66
5	0	1	1	0	1	1	0	1	6D
6	0	1	1	1	1	1	0	1	7D
7	0	0	0	0	0	1	1	1	07
8	0	1	1	1	1	1	1	1	7F
9	0	1	1	0	1	1	1	1	6F
A	0	1	1	1	0	1	1	1	77
B	0	1	1	1	1	1	0	0	7C
C	0	0	1	1	1	0	0	1	39
D	0	1	0	1	1	1	1	0	5E
E	0	1	1	1	1	0	0	1	79
F	0	1	1	1	0	0	0	1	71
P	0	1	1	1	0	0	1	1	73
不显示	0	0	0	0	0	0	0	0	00

### 10.3.2 LED显示器的显示方式

LED 显示器通常由若干个 LED 七段显示器组成，有静态显示与动态显示两种方式。

#### 1. 静态显示方式

所谓静态显示，就是当显示器显示某一个字符时，相应的发光二极管恒定地导通或截止。LED 显示器在静态显示方式下，各显示位的位选线即共阴极点（或共阳极点）连接在一起接地（或接 +5V）；各显示位的段选线（a~h）与一个 8 位并行口相连。

静态显示电路中，每一显示位可独立显示，只要在该位的段选线上保持段选码电平，该位就能保持相应的显示字符。由于每一显示位都由一个相应的 8 位输出口锁存段选码，故在同一时刻不同的显示位可以显示不同的字符。

#### 2. 动态显示方式

在多位 LED 显示时，为了简化电路，降低成本，可采用动态显示方式。所谓动态显示，就是一位一位地轮流点亮各位显示器（扫描）。对于某一位显示器来说，每隔一段时间点亮一次。显示器的亮度既与导通电流有关，也与点亮时间和间隔时间的比例有关。调整电流和时间参数，可实现亮度较高且较稳定的显示。动态显示电路中将所有显示位的段选码线并联在一起，由一个 8 位 I/O 口控制，而位选线（共阴极点或共阳极点）分别由相应的 I/O 口线控制。

8 位 LED 动态显示电路只需要两个 8 位 I/O 口，其中一个锁存段选码，另一个控制位选。由于所有显示位的段选码皆由一个 I/O 控制，因此，在每一个瞬间，8 位 LED 只可能显示相同的字符。

### 10.3.3 LED显示器接口及应用举例

从 LED 显示器的显示原理可知，为了显示字母与数字，必须最终转换成相应的段选码。这种转换可以通过硬件译码器来进行，也可以用软件进行译码。

#### 1. 硬件译码显示器接口

图 10-11 是 Motorola 公司生产的 CMOS BCD 七段十六进制锁存、译码、驱动芯片 MC14495 的逻辑引脚图。该电路的特点是可显示十六进制字符，同时还有译码器输入大于等于 10 时的指示端 (h+i)。当输入数据大于或等于 10 时，h+i 端输出“1”电平。另外还有输入数据为 15 时，电路输出端  $\overline{\text{VCR}}$  为“0”电平（其他输入状态时为高阻）的功能。电路内部还有一个  $290\Omega$  的限流电阻，故 LED 不需外加限流电阻。 $\overline{\text{LE}}$  为片选端，电路中的锁存器在  $\overline{\text{LE}}=0$  时输入数据，在  $\overline{\text{LE}}=1$  时锁存数据。

表 10-2 为 MC14495 的真值表。从表中可以看出，当显示数据大于或等于 10 时，h+i 端输出“1”电平。注意：若要显示带小数点的数据，则需要在 LED 七段显示器的 h 端另加驱动控制，MC14495 本身不能完成显示小数点的功能。

表 10-2 MC14495 的真值表

输 入	输 出	显 示
D C B A	h+i g f e d c b a	字 形
0 0 0 0	0 0 1 1 1 1 1 1	0
0 0 0 1	0 0 0 0 0 1 1 0	1
0 0 1 0	0 1 0 1 1 0 1 1	2
0 0 1 1	0 1 0 0 1 1 1 1	3
0 1 0 0	0 1 1 0 0 1 1 0	4
0 1 0 1	0 1 1 0 1 1 0 1	5
0 1 1 0	0 1 1 1 1 1 0 1	6
0 1 1 1	0 0 0 0 0 1 1 1	7
1 0 0 0	0 1 1 1 1 1 1 1	8
1 0 0 1	0 1 1 0 1 1 1 1	9
1 0 1 0	1 1 1 1 0 1 1 1	A
1 0 1 1	1 1 1 1 1 1 0 0	B
1 1 0 0	1 0 1 1 1 0 0 1	C
1 1 0 1	1 1 0 1 1 1 1 0	D
1 1 1 0	1 1 1 1 1 0 0 1	E
1 1 1 1	1 1 1 1 0 0 0 1	F

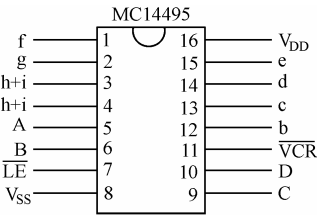


图 10-11 MC14495 逻辑引脚

#### 2. 软件译码显示器接口

由于微型计算机本身具有较强的逻辑控制能力，所以采用软件译码并不复杂。而且软件译码的译码逻辑可随意编程设定，不受硬件译码逻辑限制。采用软件译码还能简化硬件电路结构，可见，在微型计算机和单片机应用系统中，使用最广泛的还是软件译码的显示接口。

## (1) 软件译码静态显示接口

如图 10-12 所示为 CPU 通过 8255A 扩展 I/O 口控制的三位静态 LED 显示器接口的原理图。图中 LED 为共阴极接法。若为共阳极时，公共极接 +5V。如果显示位数较多。可再增加 8255A 或其他并行输出口。

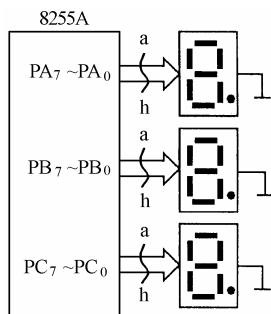


图 10-12 3 位静态 LED 显示器接口原理图

例如，某 8086 CPU 系统通过 8255A 与按键开关、LED 七段显示器等外部设备相连接，电路原理如图 10-13 所示。

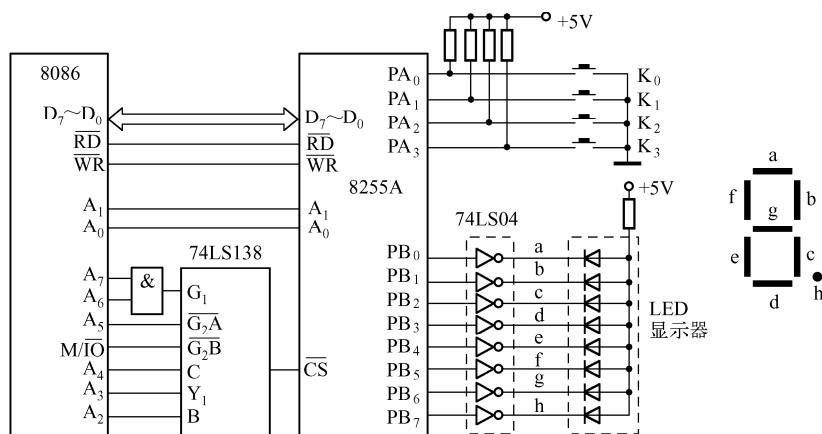


图 10-13 8255A 与按键、七段显示器连接电路原理图

由图可知，8255A 的端口 A、B、C 及控制端口的地址分别为 C4H、C5H、C6H、C7H。8255A 工作在方式 0，端口 A 输入，端口 B 输出，能够正常工作的控制字为 10010000B (90H)。电路中的 LED 七段显示器采用共阳极显示器，静态工作方式。在 LED 七段显示器段码驱动时，采用了反相驱动，所以要注意正确配置段码表。电路连接了 4 个按键开关  $K_3 \sim K_0$ ，这 4 个按键开关组成了 4 位二进制数 ( $K_3$  对应高位， $K_0$  对应低位)，并对应 1 位十六进制数。下面的程序段，实现将  $K_3 \sim K_0$  组成的 1 位十六进制数实时地在 LED 七段显示器上显示。

```
START:  MOV     AL, 90H
```

；设置方式控制字，口 A 输入，口 B 输出

	OUT	0C7H, AL	
X1:	IN	AL, 0C4H	; 输入按键状态
	AND	AL, 0FH	; 屏蔽掉不用的高4位
	MOV	BX, OFFSET LEDTAB	; 设置段码表指针
	XLAT		; 读取段码
	OUT	0C5H, AL	; 输出段码到端口 B
	MOV	AX, 200H	; 延时
X2:	DEC	AX	
	JNZ	X2	
	JMP	X1	
	HLT		
LEDTAB	DB	3FH	; 0 的段码, 设置段码表
	DB	06H	; 1 的段码
	DB	5BH	; 2 的段码
	DB	4FH	; 3 的段码
	DB	66H	; 4 的段码
	DB	6DH	; 5 的段码
	DB	7DH	; 6 的段码
	DB	07H	; 7 的段码
	DB	7FH	; 8 的段码
	DB	6FH	; 9 的段码
	DB	77H	; A 的段码
	DB	7CH	; B 的段码
	DB	39H	; C 的段码
	DB	5EH	; D 的段码
	DB	79H	; E 的段码
	DB	71H	; F 的段码

## （2）软件译码动态显示接口

如图 10-14 所示为 CPU 通过 8255A 扩展 I/O 口控制的 3 位动态 LED 显示接口。

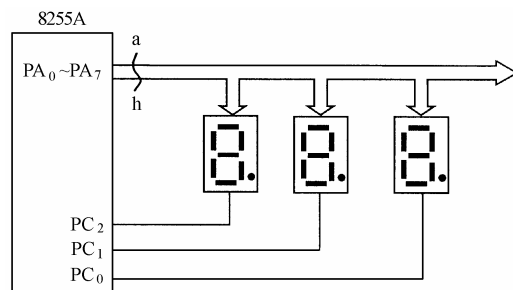


图 10-14 3 位动态 LED 显示接口原理图

动态显示程序设计中显示程序的要点：

① 解决显示译码问题。因为要显示的数字与其对应的段选码之间并没有有机的联系和转换规律，所以要用查表的方法完成这种译码功能。

② 在进入显示程序之前，为保持显示的数据，专门开辟几个单元作为显示缓冲区，用以存放要显示的数字（十六进制数）。设其首地址为 **DISMEM**，存放最左边一位显示数码，接着为 **DISMEM1**，存放左边第二位显示数码……

采用软件译码方法一般有两种表格设置方案：① 顺序表格排列法。即按一定的顺序排列显示段码。通常显示的字形数据就是该段码在段码表中的相对表头的偏移量。② 数据结构法。即按字形和段码的关系，自行设计一组数据结构。该方法设计灵活，但程序运行速度较慢。

图 10-15 为 8 位 LED 七段显示器接口电路。该电路采用共阳极 LED 显示器，为了减少所用器件的数量，该电路采用动态扫描显示方式。现选用 8255A 作为 8 位七段显示器和微处理器的接口芯片，端口 A 和 B 都用做方式 0 的输出端口，端口 A 的输出作为显示位反相驱动器的选择信号，端口 B 的输出作为段驱动器的七段代码（段码）信息。电路连接设定 8255A 端口 A、B、C 及控制端口的地址分别为 60H、61H、62H 和 63H。

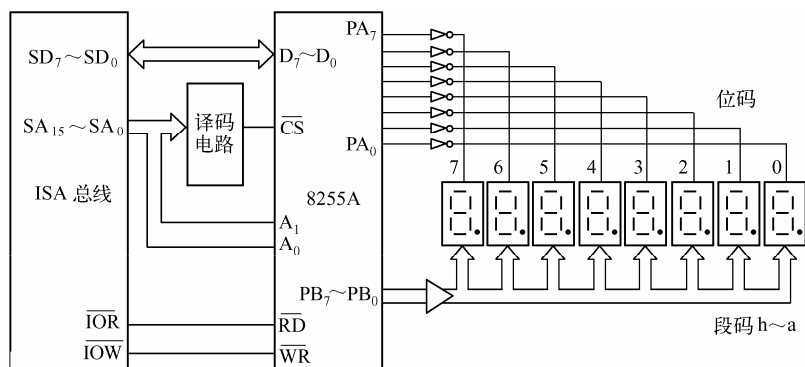


图 10-15 8 位七段 LED 显示器接口电路

下面是 8 个显示器重复显示（50 次）8 位十六进制数 13579BDF 的源程序。

DATA	SEGMENT	
TABLE:	DB 0COH	; 0 的段码，开始设置段码表
	DB 0F9H	; 1 的段码
	DB 0A4H	; 2 的段码
	DB 0BOH	; 3 的段码
	DB 99H	; 4 的段码
	DB 92H	; 5 的段码
	DB 82H	; 6 的段码
	DB 0F8H	; 7 的段码
	DB 80H	; 8 的段码
	DB 98H	; 9 的段码
	DB 88H	; A 的段码

```

        DB      83H          ; B 的段码
        DB      0C6H        ; C 的段码
        DB      0A1H        ; D 的段码
        DB      86H         ; E 的段码
        DB      8EH         ; F 的段码
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE,DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     AL, 80H
        MOV     63H, AL      ; 送各数据端口方式 0 的输出控制字
        MOV     DL, 50       ; 设置重复次数, 显示 50 次
        LEA     SI, TABLE   ; 取段码表首址
        MOV     BX, 1        ; 欲显示的字形设置为数字“1”, 是最左位显示的数
        MOV     AH, 7FH      ; 显示位 7 的位选码, 指向最左位 (第 7 显示位)
X1:     MOV     AL, [BX+SI]   ; 取数的段码, 首次取 1
        OUT     61H, AL      ; 送段选码, B 端口
        MOV     AL, AH
        OUT     60H, AL      ; 送位选码, A 端口
        ROR     AH, 1        ; 形成下一个位选码
        ADD     BX, 2        ; 形成下一个要显示的数 (奇数)
        AND     BX, 0FH
        MOV     CX, 30H      ; 延迟一定的时间, 在实际中应调整该参数
X2:     LOOP    X2
        CMP     AH, 7FH
        JNZ     X1          ; 判第 7~0 显示位是否结束
        DEC     DL
        JNZ     X1          ; 判重复显示 50 次是否结束
        MOV     AH, 4CH
        INT     21H
CODE    ENDS
        END     START
```

## 10.4 打印机及其接口

打印机是计算机最常使用的外部设备之一, 它能获得硬拷贝输出。目前计算机中广泛应用的打印机种类繁多, 但按打印机的结构基本上可分为击打式和非击打式两种。

击打式打印机印字清晰, 可以复印和长期保存, 缺点是噪声大、速度慢。目前最常使

用的击打式打印机是针式点阵打印机。针式点阵打印机是一种利用针点矩阵打印字符或图形的打印机，常用的字符点阵有  $5 \times 7$ 、 $7 \times 9$  等，汉字点阵有  $16 \times 16$ 、 $24 \times 24$  等数种。针式点阵打印机与其他类型的打印机相比，具有廉价、有复印和打印图形的能力等优点。以标准 ASCII 码送入，使用方便，字型可缩小或放大，但工作时噪声较大。

非击打式打印机由于没有打击动作，所以噪声低，省去了色带部分，但有些打印机对纸的要求较高，因而价格较高。非击打式打印机有热敏式、静电式、激光式和喷墨式等种类。

本节主要介绍微型打印机。微型打印机是指体积较小，一行打印字符小于 40 个字符的小型打印机。由于设计精巧，功能实用，常用于工业自动控制、单片机以及便携式微机和商用收款机等领域。常见的有北京工业大学电子厂的 TP $\mu$ P 系列，佳能公司的 BJ10ex 喷墨打印机等。

目前国内流行的微型打印机主要有 GP16 或 TP $\mu$ P16A，它们均以 8039 单片机（个别产品采用 8049 单片机）作为控制器的智能化微型打印机，机芯为日本生产的 Model-150-II 针打，每行可打印  $5 \times 7$  点阵的字符 16 个。

在这类微型打印机中，8039 单片机执行固化在 EPROM 中的控打程序，接收和执行主机送来的命令。通过控制口和驱动电路，实现对打印机机芯机械动作的控制。把主机送来的数据以字符串、数据或图表形式打印出来。也可以响应停机、自检、走纸等开关操作，使操作员随时对打印机状态进行干预。

这两种微型打印机的结构基本相同，所不同的是控制信号略有差别。下面介绍 TP $\mu$ P16A 微型打印机接口信号。

TP $\mu$ P16A 微型打印机与计算机应用系统通过机壳后部的 20 芯扁平电缆及接插件连接。打印机机壳后部接插件引脚信号如图 10-16 所示。

2	4	6	8	10	12	14	16	18	20
GND	GND	GND	GND	GND	GND	GND	GND	$\overline{\text{ACK}}$	$\overline{\text{ERR}}$
$\overline{\text{STB}}$	DB <sub>0</sub>	DB <sub>1</sub>	DB <sub>2</sub>	DB <sub>3</sub>	DB <sub>4</sub>	DB <sub>5</sub>	DB <sub>6</sub>	DB <sub>7</sub>	BUSY
1	3	5	7	9	11	13	15	17	19

图 10-16 微型打印机接口信号

DB<sub>7</sub>~DB<sub>0</sub> 为数据传送线，单向，由计算机输入打印机。

$\overline{\text{STB}}$  为数据选通输入信号。在此信号的上升沿时，数据线上的 8 位并行数据被打印机读入机内锁存。

BUSY 为打印机“忙”状态信号，输出，高电平有效。当此信号有效时，表示打印机正忙于处理上一个数据，此时，主计算机不得使用  $\overline{\text{STB}}$  信号向打印机送入新的数据。

$\overline{\text{ACK}}$  为打印机的应答信号，输出，低电平有效。当此信号有效时，表明打印机已取走数据线上的数据。

$\overline{\text{ERR}}$  为出错信号，输出。当送入打印机的命令格式有错时，打印机立即打印出一行出错信息，以提示操作者注意。在打印机打印出错信息之前，该信号线上出现一个负脉冲，脉冲宽度 30ms。



TPμP16A 微型打印机在使用时，扁平电缆的长度应小于 5m，并应尽可能地短，以保证控制信息和数据信息的正确传送。

在简单的应用中，一般连接  $\overline{DB_7} \sim \overline{DB_0}$ 、 $\overline{STB}$ 、BUSY 信号即可使用。

【例 10-1】微型打印机的接口与编程。某微型打印机通过 8255A 与 8088 CPU 连接，接口电路如图 10-17 所示。试编程实现将从 DATA 开始的 100 字节的 ASCII 码字符数据送打印机打印的程序。

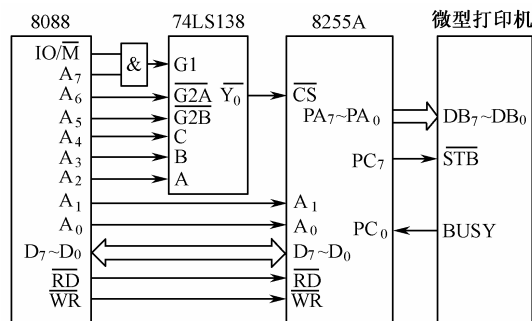


图 10-17 微型打印机接口电路

由图可知，可编程接口芯片 8255A 的端口地址，从 A 口至控制口分别为 80H~83H。A 口为工作方式 0 输出，C 口为高半字节输出，C 口为低半字节输入，方式控制字设定为 10001000B。

按题目要求，编程如下：

```

START:  MOV  AL, 88H
        OUT  83H, AL
        MOV  AL, 0FFH
        OUT  82H, AL
        MOV  SI, OFFSET DATA
        MOV  CX, 100
X1:     IN   AL, 82H
        TEST AL, 01H
        JNZ  X1
        MOV  AL, [SI]
        INC  SI
        OUT  80H, AL
        MOV  AL, 00H
        OUT  82H, AL
        MOV  AL, 0FFH
        OUT  82H, AL
        LOOP X1
        HLT
DATA    DB   x1, x2, x3, x4, ..., x100

```

## 10.5 视频系统

视频系统是计算机系统的重要组成部分，包括显示器和显示适配器（通常被称为显示控制卡、显示卡、显卡）。用户主要是通过键盘和显示器进行人机对话来实现计算机操作。没有显示器用户就无法进行工作。显示器能将计算机的输出信息转换成各种直观的图形、图像和字符，程序、数据也能在屏幕上显示出来。它具有速度快、无噪声、无机械磨损、直观形象、方便可靠等优点，在国防、航天、人工智能等高科技领域，以及办公自动化等方面的应用越来越广。由于显示器屏幕上的信息只能供观察，不能永久地记录，所以又称为软拷贝设备。

显示器的种类很多，按使用的显示器件不同，主要分为两大类：一类是阴极射线管显示器，简称为 CRT 显示器；另一类是平板显示器，主要包括发光二极管显示器、液晶显示器和等离子体显示器等。

### 10.5.1 CRT显示器

#### 1. CRT显示器的基本组成

CRT 显示器的基本组成主要包括视频放大驱动电路、行扫描电路、场扫描电路、高压电路、CRT 显像管以及机内直流电源等六大部分，如图 10-18 所示。

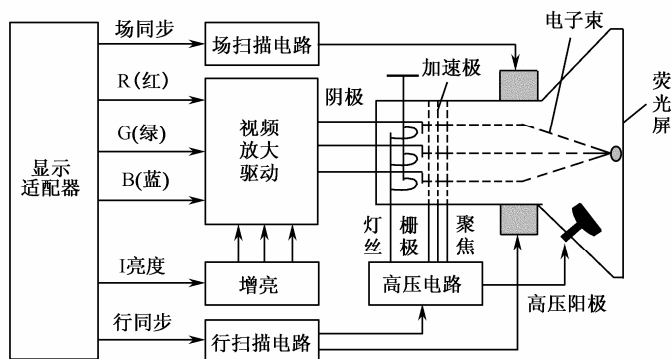


图 10-18 CRT 显示器组成框图

在图 10-8 中，视频放大驱动电路将主机经显示卡送来的视频信号放大驱动后，送到 CRT 显像管的阴极，产生电子束轰击屏幕而出现光点。由显示卡送来的水平（行）同步信号与垂直（场）同步信号，分别经行扫描电路和场扫描电路，为 CRT 显像管的水平（行）偏转线圈和垂直（场）偏转线圈提供具有一定幅度和线性良好的锯齿波电流，产生垂直方向和水平方向的偏转磁场，控制电子束在水平或垂直方向的偏转，形成扫描光栅。

CRT 显示器的显像管是用来实现字符或图形显示的电-光转换器件，要使其正常显示，必须按要求为它的各个电极提供所需的高、中、低电压。高压电路就是用来产生这些电压的部分，它利用行扫描电路产生的行逆程脉冲，经过行逆程变压器（俗称高压包或行输出变压

器) 升压、整流、滤波后, 为 CRT 显像管提供正常工作所需的各挡电压, 以及为视频等电路提供电源电压, 以便显像管正常显示光栅或图形、字符。

显示器内的直流电源一般均采用开关稳压电源, 通过电路对交流电网电压进行变换, 输出几挡直流电压, 为显示器内部各电路提供所需的直流电压。一般来讲, TTL 数字集成电路需要 +5V 直流电压, 视频信号处理及行、场振荡需要 +12V 直流电压, 场输出需要 +25V 左右的直流电压, 视放级需要 +80~+150V 不等的直流电压, 行输出需要几十伏至一百多伏不等的直流电压。

CRT 显像管的结构很简单, 它是一个真空玻璃管, 在其颈部有一个电子枪, 电子枪的对面有一个荧光敷面。当电子枪被激活时, 即发射一串电子束, 电子束轰击屏幕内侧的荧光敷面, 从而在屏幕上产生一个荧光点。真空管上附有水平偏转线圈和垂直偏转线圈, 依据电磁吸引、排斥原则形成电子束的偏转, 控制屏幕表面的电子束的位置。对光栅扫描方式而言, 不同的信号作用于水平、垂直线圈, 从而使电子束在屏幕上移动。随着电子束的移动, 在屏幕上就会留下一条荧光轨迹, 这条轨迹经过一段时间才能消散。这段时间是由荧光敷面的特性决定的, 荧光消散率由荧光暂留时间决定。

## 2. 光栅扫描

电子束首先出现在屏幕的左上角, 再扫至右上角, 这样在屏幕上就留下一条线。这条线称为光栅线, 当电子束到达屏幕右边后, 它又以极高的速度重新定位到屏幕左边, 这样在

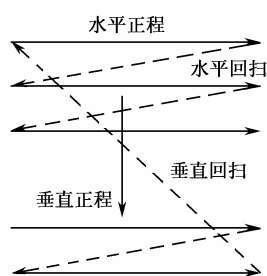


图 10-19 光栅扫描原理

第一条轨迹的下面又出现一条线, 叫水平回扫。从这点开始经水平扫描又在屏幕上产生第二条显示线。这个扫描过程一直持续到水平扫描到达屏幕底部, 此时一帧画面扫描完毕。最后, 电子束又以极高的速度回扫到屏幕左上角, 称为垂直回扫。扫描原理如图 10-19 所示。在水平回扫和垂直回扫的过程中, 要对电子束进行抑制 (消隐), 使回扫线不在屏幕上显示出来。于是在屏幕上就出现了一条条水平的光栅扫描线。

为了避免人的眼睛产生闪烁的感觉, 帧扫描必须每秒至少进行 50 次。

电子束轰击荧光屏产生发光点, 由于人眼的视觉暂留特性, 各个发光点看起来组成了图像。对于彩色显示器, 像素点的颜色由 R、G、B 三种基本颜色按一定比例进行配色。三色荧光点由红、绿、蓝三种颜色的 3 个小荧光粉点组成, 排成三角形。一般 CRT 显示器采用荫罩式显像管。荫罩是一个类似筛子的网罩, 网罩的每一个网眼对应一个三色荧光点, 有 R、G、B 三个电子枪, 穿过网眼打在对应的小荧光粉点上。

## 10.5.2 液晶显示器

液晶显示器从结构上说, 属于平板显示器件, 简称 LCD (Liquid Crystal Display) 显示器。

### 1. 液晶及其分类

LCD 显示器的基础材料是液晶。从字面看, 液晶就是“液态”的“晶体”, 可见, 它具有液体的“流动性”, 又具有晶体的“光学各向异性”。奥地利生物学家 F.Reinitzer 在

1888 年首先观察到液晶现象，经过进一步确认后，将这种“兼有液体流动性和晶体光学各向异性的液体”，建议称之为“液晶”。可见，液晶是一种介于固态和液态之间的物质，是具有规则性分子排列的有机化合物，如果把它加热会呈现透明状的液体状态，把它冷却则会呈现结晶颗粒的混浊固体状态。液晶态物质既具有液体的流动性和连续性，又保留了晶体的有序排列性，物理上呈现各向异性。

液晶显示器的基本原理：在电场作用下，液晶分子从特定的初始排列状态转变为其他分子排列状态，随着分子排列的变化，液晶的光学特性发生变化，从而产生视觉的变化。

从液晶的物理条件角度，液晶可以分为热致液晶和溶致液晶两大类。目前用于显示的液晶材料基本上都是热致液晶。热致液晶是指当液晶物质加热时，在某一温度范围内呈现出各向异性的液体。用于显示的都是可工作在室温的热致液晶。

从分子结构角度，液晶可分为层列液晶、丝状液晶和脂状液晶三种。液晶显示器使用的是类似细火柴棒的丝状液晶。

按物理结构，常见的液晶显示器分为 TN-LCD（Twisted Nematic-LCD，扭曲向列 LCD）、STN-LCD（Super TN-LCD，超扭曲向列 LCD）、DSTN-LCD（Double layer STN-LCD，双层超扭曲向列 LCD）和 TFT-LCD（Thin Film Transistor-LCD，薄膜晶体管 LCD）四种。

TN 型是液晶显示器中最基本的显示技术，之后其他种类的液晶显示器是在 TN 型基础上改良的。TN 型的工作原理较其他技术更简单。

STN 型的显示原理与 TN 型相类似。不同在于，TN 型的液晶分子是将入射光旋转  $90^\circ$ ，而 STN 型是将入射光旋转  $180^\circ \sim 270^\circ$ 。

DSTN 型是通过双扫描方式来扫描扭曲向列型液晶显示屏，从而达到完成显示的目的。DSTN 是由 STN 型发展而来的。由于 DSTN 型采用了双扫描技术，因此显示效果相对 STN 型来说，有大幅度提高。

TFT 型液晶显示器较为复杂，与 TN 型液晶显示器相比，采用了不同的显示方式，必须利用背光源，是目前主流液晶显示器的面板。

## 2. TN 型液晶显示器的原理

TN 型的液晶显示器已基本被淘汰。

如图 10-20 所示。TN 型液晶显示器的基本构造为上下两片导电玻璃基板，其间注入向列型液晶，上下基板外侧各加上一片偏光板。另外在导电膜上涂一层具有极细沟纹的配向膜。由于液晶分子拥有液体的流动特性，所以很容易顺着沟纹方向排列。上下基板的沟纹方向以  $90^\circ$  垂直配置。当液晶填入后，由于接近基板沟纹的束缚力较大，所以液晶分子会沿着上下基板沟纹方向排列，而中间部分的液晶分子束缚力较小，会形成扭转排列。因为使用的液晶是向列型的液晶，且液晶分子扭转  $90^\circ$ ，故称为 TN 型。

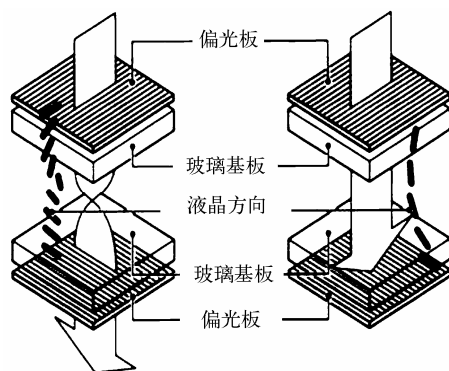


图 10-20 TN 型液晶显示屏结构

若不施加电压，则进入液晶组件的光会随着液晶分子扭转方向前进，由于上下两片偏光板和配向膜同向，所以光可通过，并形成亮的状态；若施加电压，则液晶分子朝施加电场的方向排列，垂直于配向膜，因此光无法通过第二片偏光板，形成暗的状态。这样会形成透光时为白、不透光时为黑，字符就可以显示在屏幕上了。

### 3. TFT型液晶显示器的原理

TFT 型液晶显示器与 TN 型液晶显示器的原理不同，但在构造上两者仍有相似之处，如玻璃基板、配向膜、偏光板等，也同样采用两夹层间填充液晶分子的设计，只不过把 TN 上部夹层的电极改为 FET 晶体管，而下层改为共同电极。在光源设计上，TFT 型的显示采用“背透式”照射方式，即假想的光源路径不是像 TN 型液晶那样从上至下，而是从下向上，这样的做法是在液晶的背部设置类似日光灯的光管，光源照射时通过下偏光板向上透出。在 FET 的电极导通时，液晶分子的表现如 TN 型液晶的排列状态一样会发生改变，通过遮光和透光来达到显示的目的。

不同的是，由于 FET 型晶体管具有电容效应，能够保持电位状态，先前透光的液晶分子会一直保持这种状态，直到 FET 电极下一次再加电改变其排列方式。相对而言，TN 型就没有这个特性，液晶分子一旦没有施压，就立刻返回原始状态。这是 TFT 型液晶和 TN 型液晶显示的最大不同之处。

### 4. 液晶显示的驱动

液晶显示的驱动就是调整施加在液晶显示器件电极上的电位信号的相位、峰值、频率等，建立驱动电场。液晶显示的驱动方式有许多种，常用的驱动方法有静态驱动法和动态驱动法。TN 型及 STN-LCD 型一般采用静态驱动法。

#### （1）静态驱动法

静态驱动法是最基本的方法。它适用于笔段型液晶显示器件的驱动。当多位数字组合时，各位的背电极连接在一起。振荡器的脉冲信号经分频后直接施加在液晶显示器件的背电极上。段电极的脉冲信号由显示选择信号与时序脉冲通过逻辑异或合成产生。当某位显示像素被显示选择时，该显示像素上两电极的脉冲电压相位相差  $180^\circ$ ，在显示像素上产生 2V 的电压脉冲序列，使该显示像素呈现显示特性。当某位显示像素为非显示选择时，该显示像素上两电极的脉冲电压相位相同，在显示像素上合成电压脉冲为 0V，从而实现不显示的效果。为提高显示的对比度，适当地调整脉冲的电压即可。

#### （2）动态驱动法

当液晶显示器件上显示像素众多时，如点阵型液晶显示器件，为了简化硬件驱动电路，在液晶显示器件电极的制作与排列上做了加工，实施了矩阵型的结构，即把水平一组显示像素的背电极都连在一起引出，称之为行电极，把纵向一组显示像素的段电极都连接起来一起引出，称之为列电极。液晶显示器上的每一个显示像素都由其所在的列与行的位置唯一确定。在驱动方式上采用类同于 CRT 的光栅扫描方法。液晶显示的动态驱动法循环地给行电极施加选择脉冲，同时按所显示的数据，在列电极上给出相应的选择或非选择的驱动脉冲，从而实现某行所有显示像素的显示功能。这种行扫描是逐行顺序进行的，循环周期很短，使得液晶显示屏上呈现出稳定的图像。

在液晶显示的驱动中，还涉及一个驱动电压极性变换的问题。液晶分子有一个特性，

就是不能够固定在某一个电压值不变。如果固定在某一个电压值太久，那么即使这个电压取消，液晶分子也会因为特性被破坏而无法随电场的变化而变化。所以，每隔一段时间，就必须改变电压大小，防止液晶分子的特性被破坏。

驱动电压分为两种极性：正极性和负极性。当驱动电位高于零电位时，称为正极性。当驱动电位低于零电位时，称为负极性。不管是正极性还是负极性，只要电压的绝对值相同，就会得到相同的亮度。但是，在两种极性情况下，液晶分子的转向是完全相反的，这就避免了液晶分子长时间固定在一个方向造成特性被破坏的情况。

### 10.5.3 字符和图形显示的基本原理

#### 1. 字符显示方法

为了叙述方便，以黑白字符显示为例来介绍 CRT 显示器字符显示的工作原理。图 10-21 是黑白字符显示的原理示意图。它由定时控制电路、显示存储器 VRAM、字符发生器 ROM、移位寄存器和视频信号合成电路组成。

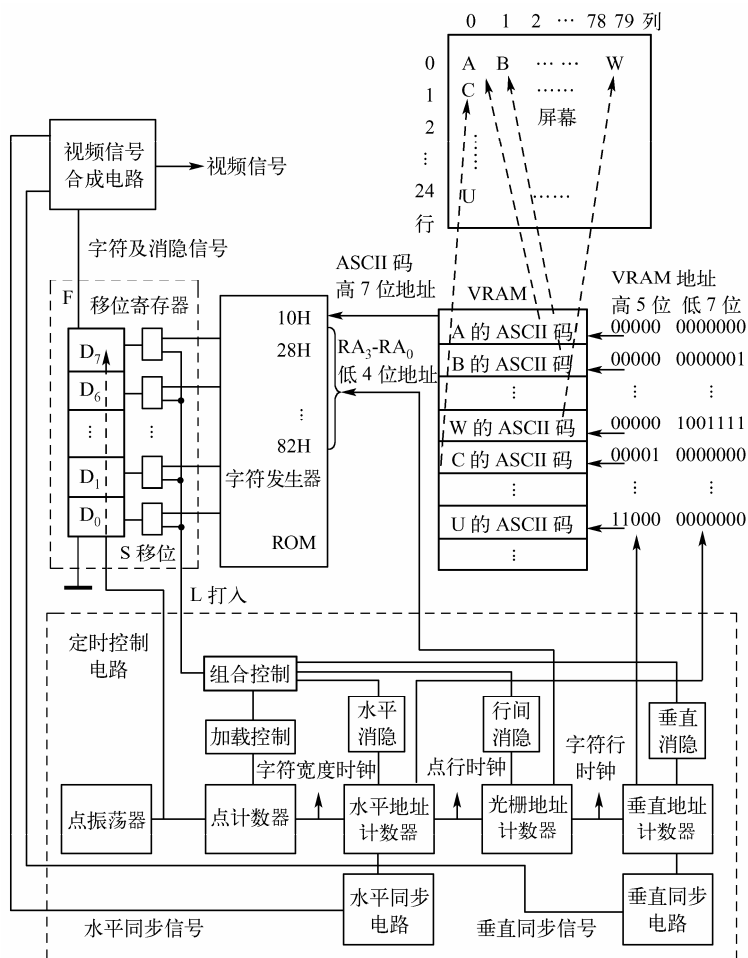


图 10-21 黑白字符显示工作原理示意图

在 CRT 显示器进行光栅扫描的过程中，当电子束进行水平正程扫描时，图像信号通过控制电子束来控制各点的亮度，以便在 CRT 屏幕上形成图像。而在字符显示的情况下，只要用视频信号对扫描电子束简单地进行“开”或“关”控制，就可以用点阵在屏幕上组成字符。当要在屏幕上显示字符时，通常采用的方法是把有效的显示屏幕划分成许多方块。每个方块称为字符窗口，要显示的字符就位于字符窗口中。对应于每个字符窗口，所要显示的字符代码被存放在称为视频随机存储器的 VRAM 中。字符窗口是指每个字符在屏幕上所占的点数，它包括字符显示点阵和字符间隔。在 IBM PC 系统中，屏幕上共显示 80 列×25 行=2000 个字符，故字符窗口数目为 2000。在单色字符方式下，每个字符窗口为 9×14 点阵，字符为 7×9 点阵。

由图 10-21 可以看出，对于字符显示器来说，显示存储器 VRAM 中存放的是 ASCII 字符代码，它保存显示屏幕某个字符窗口中要显示的某个字符。VRAM 的单元与屏幕上显示的字符（有 80 列×25 行个字符）一一对应，即由首地址开始每 80 个单元的内容对应屏幕上一行 80 个字符，每 80 个单元中的首地址单元的内容对应屏幕每行的第 0 列，每 80 个单元中的末地址单元的内容对应屏幕每行的第 79 列。

垂直地址计数器和水平地址计数器的输出就是 VRAM 的地址。VRAM 中的高 5 位地址由垂直地址计数器给出，它指出每 80 个单元的高 5 位地址，即屏幕 0~24 行的行地址。VRAM 中的低 7 位地址由水平地址计数器给出，它指出每 80 个单元的低 7 位地址，即屏幕 0~79 列的列地址。

ASCII 字符图形的点阵是由字符发生器 ROM 产生的。每个 ASCII 字符图形的 9 字节连续存放在字符发生器 ROM 中。ROM 的高位地址是来自 VRAM 中的 ASCII 代码。换言之，ASCII 字符代码作为字符发生器中这组字符点阵字节的高位地址，它具体指出要选择哪一组字符点阵；而 ROM 的低位地址则来自光栅地址计数器的内容，它具体指出这组字符点阵字节中的哪一字节，即图 10-22 中 0000B~1000B 行中的哪一行。

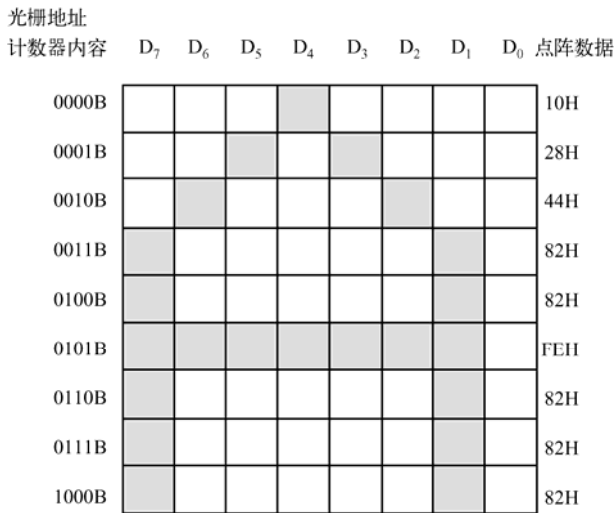


图 10-22 字符 A 的点阵图形

在 CRT 进行光栅扫描的过程中,从字符发生器中依次读出某个字符的点阵字节,送入 8 位移位寄存器。移位寄存器的引脚 L 为并行输入控制端,高电平有效。移位寄存器的引脚 S 为移位控制端。字符的点阵字节经移位寄存器逐位输出,按照点阵中 0 和 1 代码的不同控制扫描电子束的开或关,从而在屏幕上显示出字符图形。应注意的是,8 位移位寄存器 D<sub>0</sub> 位的移位输入端接地,每次打入后,移位超过 8 次时,均移出 0。

CRT 显示器的逐点、逐字、逐行、逐屏的刷新显示,是由定时控制电路中的点计数器、水平地址计数器、光栅地址计数器和垂直地址计数器来控制实现的。

点振荡器产生 16.257MHz 的点时钟,用来控制移位寄存器依次移位输出逐点对应的 0 或 1 信号。点计数器对点时钟 9 分频,输出字符时钟定时信号。也就是说,1 个字符时钟周期对应 9 个点时钟周期,其中 7 个点时钟周期对应显示字符点阵的 0 或 1 信号,2 个点时钟周期对应显示字符列间隔的 00 信号。只有模 9 点计数器为 0000 时,才控制移位寄存器的加载。

水平地址计数器对一行显示字符进行控制,送出当前要显示的这一行字符的 VRAM 地址,每行有效显示 80 个字符。当计数到 80 时产生高电平水平回扫消隐信号,使移位寄存器打入信号 L 为 0,即控制移位寄存器不要加载。水平消隐共占用 18 个字符时钟周期。当计数到 82 时产生高电平水平同步信号。

光栅地址计数器对字符窗口的高度进行控制,因字符窗口的高度为 14,字符点阵高度为 9,行间隔为 5 个点,故光栅地址计数器控制一个字符的 9 行点阵逐行输出,当光栅地址计数器计数到 9 时产生高电平行间消隐信号,控制移位寄存器打入信号 L 为 0,即控制移位寄存器不要加载,使最后 5 行进行行间消隐。

垂直地址计数器控制屏幕 25 行字符的显示,与水平回扫类似,光栅到达屏幕底部最后一行字符时需要垂直回扫。当垂直地址计数器计数到 25 时产生高电平垂直回扫消隐信号和高电平垂直同步信号。垂直消隐信号控制移位寄存器打入信号 L 为 0,即控制移位寄存器不要加载。垂直消隐共占用一行字符的显示时间。

了解了黑白字符显示部件的功能及作用后,对黑白字符显示的工作过程概述如下:

(1) 初始态,定时控制电路中的点计数器、水平地址计数器、光栅地址计数器和垂直地址计数器计数值均为 0。此时,垂直地址计数器给出 VRAM 中对应屏幕第 0 行字符的地址。水平地址计数器给出 VRAM 中对应屏幕第 0 列字符的地址。VRAM 向字符发生器 ROM 输出对应屏幕第 0 行、第 0 列要显示字符的 ASCII 码,该 ASCII 码向字符发生器 ROM 指出要显示字符窗口的高 7 位地址(即哪一组字符点阵)。光栅地址计数器给出字符发生器 ROM 中对应屏幕第 0 行、第 0 列要显示字符窗口的第 0 行字节的地址。字符发生器 ROM 向移位寄存器输出对应屏幕第 0 行、第 0 列要显示字符窗口的第 0 行字节值。点计数器向移位寄存器输出打入信号 L,将对应屏幕第 0 行、第 0 列要显示字符窗口的第 0 行字节值打入移位寄存器中。点时钟控制移位寄存器依次移位输出逐点对应的 0 或 1 信号。视频信号合成电路向阴极射线管的阴极输出视频信号,在 (X, Y) 偏转线圈施加偏转电压的控制下,在屏幕的左上角显示出第 0 行、第 0 列字符窗口的第 0 行字节值所对应的图形点。

(2) 当模 9 点计数器回 0 时向水平地址计数器输入计数脉冲,水平地址计数器加 1。此时水平地址计数器计数值为 1,而点计数器、光栅地址计数器和垂直地址计数器计数值均



为 0, 则将在屏幕上显示出第 0 行、第 1 列字符窗口的第 0 行字节值所对应的图形点。当模 9 点计数器又回 0 时向水平地址计数器输入计数脉冲, 水平地址计数器加 1。因此时水平地址计数器计数值为 2, 而点计数器、光栅地址计数器和垂直地址计数器计数值均为 0, 则将在屏幕上显示出第 0 行、第 2 列字符窗口的第 0 行字节值所对应的图形点。重复上述过程, 直至水平地址计数器计数值为 79, 而点计数器、光栅地址计数器和垂直地址计数器计数值仍均为 0, 则将在屏幕上显示出第 0 行、第 79 列字符窗口的第 0 行字节值所对应的图形点。该过程就是屏幕中第 0 条水平扫描线正程的工作过程。

(3) 当水平地址计数器计数值为 80~97 期间, 水平消隐控制移位寄存器不接收字符发生器 ROM 输出的内容, 使移位寄存器输出均为 0, 即不显示亮点。当水平地址计数器计数值为 82 时产生高电平水平同步信号, 使屏幕中第 0 条水平扫描线开始逆程扫描。

(4) 当模 98 水平地址计数器回 0 时向光栅地址计数器输入计数脉冲, 光栅地址计数器加 1。因此时光栅地址计数器计数值为 1, 而点计数器、水平地址计数器和垂直地址计数器计数值均为 0, 则将在屏幕上显示出第 0 行、第 0 列字符窗口的第 1 行字节值所对应的图形点。当模 98 水平地址计数器又回 0 时向光栅地址计数器输入计数脉冲, 光栅地址计数器加 1。因此时光栅地址计数器计数值为 2, 而点计数器、水平地址计数器和垂直地址计数器计数值均为 0, 则将在屏幕上显示出第 0 行、第 0 列字符窗口的第 2 行字节值所对应的图形点。重复上述过程, 直至光栅地址计数器计数值为 8, 而点计数器、水平地址计数器和垂直地址计数器计数值仍均为 0, 则将在屏幕上显示出第 0 行、第 0 列字符窗口的第 8 行字节值所对应的图形点。当模 98 水平地址计数器计数为 79 时, 则将在屏幕上显示出第 0 行、第 79 列字符窗口的第 8 行字节值所对应的图形点。当模 98 水平地址计数器又回 0 时, 则将在屏幕上显示出第 0 行的 80 个字符。该过程就是屏幕中第 0 行的 80 个字符显示的工作过程。

(5) 当光栅地址计数器计数值为 9~13 期间, 行间消隐控制移位寄存器不接收字符发生器 ROM 输出的内容, 使移位寄存器输出均为 0, 即不显示亮点。该过程就是在屏幕中第 0 行字符与第 1 行字符之间产生 5 条消隐扫描线作为字符行间间隔的工作过程。

(6) 当模 14 光栅地址计数器回 0 时向垂直地址计数器输入计数脉冲, 垂直地址计数器加 1。因此时垂直地址计数器计数值为 1, 而点计数器、水平地址计数器和光栅地址计数器计数值均为 0, 则将在屏幕上显示出第 1 行、第 0 列字符窗口的第 0 行字节值所对应的图形点。当模 14 光栅地址计数器回 0 时向垂直地址计数器输入计数脉冲, 垂直地址计数器加 1。因此时垂直地址计数器计数值为 2, 而点计数器、水平地址计数器和光栅地址计数器计数值均为 0, 则将在屏幕上显示出第 2 行、第 0 列字符窗口的第 0 行字节值所对应的图形点。重复上述过程, 直至垂直地址计数器计数值为 24, 而点计数器、水平地址计数器和光栅地址计数器计数值仍均为 0, 则将在屏幕上显示出第 24 行字符及行间间隔所对应的图形点。该过程就是屏幕中 25 行字符显示的工作过程。

(7) 当垂直地址计数器计数值为 25 时产生垂直消隐信号和垂直同步信号, 垂直消隐控制移位寄存器不接收字符发生器 ROM 输出的内容, 使移位寄存器输出均为 0, 即不显示亮点。垂直同步信号使光栅垂直回扫。该过程就是屏幕中 25 行字符显示结束又回到初始态 1 的工作过程。

在字符显示部件中, 显示存储器 VRAM 具有重要的作用, 其内容和 CRT 显示画面是相

互对应的。首先，CPU 把要在屏幕上显示的内容存入视频存储器 VRAM，然后在 CRT 控制器的控制下，不断从 VRAM 中读出已存入的信息，再将该信息转换成所需的视频信号，送到 CRT 的阴极去控制电子束，以便在 CRT 上显示稳定的字符和图像。

在 VRAM 中，每一个字符占用 2 字节，其中，1 字节是字符的 ASCII 码，1 字节是其属性码，表示字符前景色和背景色及闪烁、高亮度、反显等显示特征。ASCII 码字节一定存放在偶地址单元，属性字节一定存放在奇地址单元。若屏幕格式为 80 列×25 行，共有 2000 字符，则需要 2KB 的字符缓冲器存放要显示的字符代码和 2KB 的属性缓冲器存放要显示的字符属性代码。VRAM 尽管是在显示器接口板上的，但却是和主机的内存统一编址的。VRAM 是一个双端口存储器，既能服务于视频适配器，又能被主机 CPU 访问。不同的视频适配器其 VRAM 的大小也可能不同。

2. 图形显示方法

显示图像的最小单位是像素，而像素又以色彩为数据描述对象。如图 10-23 所示为 3 种典型的在不同颜色数下的像素组合形式。

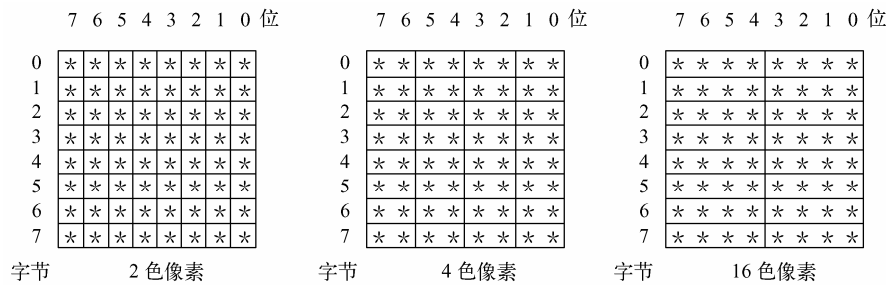


图 10-23 3 种典型的在不同颜色数下的像素组合

如果分辨率为 320×200，则共计有 64 000 个像素点。若用 1 位二进制数代表一个像素，那么这种模式最多可描述 2 种颜色，8 个像素点占用 1 字节，总共占用 64 000/8=8000 字节的 VRAM。若用 2 位二进制数代表一个像素，那么这种模式最多可描述 4 种颜色，4 个像素点占用 1 字节，总共占用 64 000/4=16 000 字节的 VRAM。同理，若用 4 位二进制数代表一个像素，那么这种模式最多可描述 16 种颜色，2 个像素点占用 1 字节，总共占用 64000/2=32 000 字节的 VRAM。可见，在图形方式下，分辨率和像素模式决定了 VRAM 的大小。像素点越多，VRAM 越大。颜色数越多，VRAM 越大。

10.5.4 显示器的主要性能指标

1. 像素和点距

构成图像的最小单位或构成图像的点叫像素。

点距是显示屏上像素之间的最小距离。这个距离不能用软件来更改，这一点与分辨率是不同的。可以通过点距直接计算显示器的最大分辨率（用显示区域的宽和高分别除以点距，即得到显示器在水平和垂直方向可以显示的最高点数）。点距越小，像素密度越大（制造越困难），显示出来的图像越细腻，越清晰。早期的 14 英寸显示器分为 0.28、0.31、0.39mm 几种规格，其中前者一直沿用，后两者已经被淘汰。目前高清晰大屏幕显示器通常

采用 0.28、0.27、0.26、0.25mm 的点距，有些产品甚至达到 0.21mm。

CRT 的点距会因为荫罩或光栅的设计、显卡的种类、垂直或水平扫描频率的不同而有所改变，而 LCD 显示器的像素数量是固定的，因此在尺寸与分辨率都相同的情况下，大多数液晶显示器的像素间距基本相同。分辨率为  $1024 \times 768$  的 15 英寸 LCD 显示器，其像素间距均为 0.297mm，而 17 英寸的基本都为 0.264mm。一般认为每个像素在 0.25~0.30mm 之间人眼会觉得比较舒服，所以液晶显示器的标准点距都在这一范围。

## 2. 分辨率

分辨率是指整屏可显示的像素的多少。最大分辨率与屏幕尺寸和点距密切相关。例如，15 英寸的显示器，当点距是 0.28mm 时，就可具有  $1024 \times 768$  点的最大分辨率。在相同分辨率下，点距越小，图像就越清晰。

分辨率通常以乘法形式表示，比如  $1024 \times 768$ ，其中“1024”表示屏幕上水平方向显示的点数，“768”表示垂直方向的点数。可见，所谓分辨率就是指画面的解析度，由像素构成，像素数越多，其分辨率就越高。

分辨率的概念与图像的像素也有直接的关系，比如，一张分辨率为  $640 \times 480$  的图片，其分辨率可达到 307 200 像素，这就是常说的 30 万像素，而一张分辨率为  $1600 \times 1200$  的图片，其像素为 200 万。可见，这里的分辨率的两个数字表示的是图片在宽、高上占的点数的单位。

对 LCD 液晶显示器和 CRT 显示器而言，分辨率都是重要的参数之一。CRT 显示器所支持的分辨率较有弹性，而 LCD 的像素点距已经固定，所以支持的显示模式不像 CRT 那么多。LCD 的最佳分辨率，也叫最大分辨率，只有在该分辨率下，液晶显示器才能显现最佳图像。

目前 15 英寸 LCD 的最佳分辨率为  $1024 \times 768$ ，17~19 英寸的最佳分辨率通常为  $1280 \times 1024$ ，更大尺寸拥有更大的最佳分辨率。

## 3. 显示器的尺寸

CRT 显示器的尺寸指显像管的对角线尺寸。最大可视面积就是显示器可以显示图形的最大范围。显像管的大小通常以对角线的长度来衡量，以英寸为单位（1 英寸=2.54cm），常见的有 15 英寸、17 英寸、19 英寸等。显示屏的水平方向与垂直方向之比一般为 4:3。15 英寸显示器的可视范围在 13.8 英寸左右，17 英寸显示器的可视区域大多在 15~16 英寸之间，19 英寸显示器的可视区域达到 18 英寸左右。

LCD 显示器的尺寸是指液晶面板的对角线尺寸，以英寸为单位，可视面积较大，15 英寸的 LCD 显示器的实际显示面积相当于 17 英寸的 CRT 显示器，主流的尺寸有 15 英寸、17 英寸、19 英寸等。

## 4. 扫描方式

CRT 显示器的扫描方式分为“逐行扫描”和“隔行扫描”两种。隔行扫描是每隔一行显示一行，到底后再返回显示刚才未显示的行，而逐行扫描是顺序显示每一行。逐行扫描比隔行扫描具有更稳定的显示效果。目前，只有家用电视仍然采用隔行扫描方式。

液晶显示器为了兼容信号源，也采用扫描方式。但这种扫描方式与 CRT 不同，并没有扫描线存在。

## 5. 像素的颜色范围

一个像素可显示出多少种颜色，由表示这个像素的二进制位数决定（又称像素的位宽），如果每个像素使用 8b（1 字节）来表示它的颜色，则每个像素可有 256 色。如使用 16b，有 65 536 种颜色；如用 24b 表示，则可有 16.8 兆种颜色。

那么如何根据显示存储器中像素值来生成每个像素的不同颜色呢？在显示适配器内有数模转换器 DAC，它能将二进制整数转换成对应亮度（灰度）的 R、G、B 模拟信号，这三个模拟信号被送到 CRT，轰击三色荧光点，从而使每个像素发出所需要的颜色。

## 6. 刷新频率

刷新频率就是屏幕刷新的速度。刷新频率越低，图像闪烁和抖动得就越厉害，眼睛疲劳得就越快。当采用 70Hz 以上的刷新频率时可基本消除闪烁。因此，70Hz 的刷新频率是显示器稳定工作的最低要求。如能达到 80Hz 以上的刷新频率就可完全消除图像闪烁和抖动感，眼睛也不会太容易疲劳。虽然刷新频率越高越好，但是过高的刷新频率会加速 CRT 显像管的老化，通常 85Hz 比较合适。

液晶显示器的发光原理与 CRT 不同。液晶显示器每一个点在收到信号后就一直保持发光状态，不像 CRT 显示器那样需要不断刷新。因此，液晶显示器画质高而且不会闪烁，把眼睛的疲劳感降到了最低。所以液晶显示器并不需要太高的刷新频率，在 60Hz 的刷新频率时画面的显示效果是比较好的。

## 7. 视频带宽

视频带宽是造成显示器性能差异的一个比较重要的因素。带宽决定着一台显示器可以处理的信息范围，即特定电子装置能处理的频率范围。工作频率范围早在电路设计时就已经被限定下来了，由于高频会产生辐射，因此高频处理电路的设计更为困难，成本也高得多。而增强高频处理能力可以使图像更清晰。所以，高带宽能处理的频率更高，图像也更好。每种分辨率都对应着一个最小可接受的带宽。

一般来说，可接受带宽的公式为

$$\text{可接受带宽} = \text{水平像素} \times \text{垂直像素} \times \text{刷新频率} \times 1.344$$

## 10.5.5 显示适配器

显示适配器又称显示卡、显卡、图像加速卡等，通常以附加卡的形式安装在主板的扩展槽中或集成在主板上。显示适配器是主机 CPU 与显示器之间的接口，其作用是接受 CPU 的命令、访问显示存储器 VRAM、产生屏幕所需的信号。

PC 采用了许多不同的视频显示标准，从 MDA、CGA、EGA 到 VGA 等，每一种视频标准都有相应的显示适配器与之对应。几种视频标准的特点如下。

**MDA (Monochrome Display Adapter):** 单色字符显示适配器。MDA 是 PC 最早使用的显示标准。采用 9×14 点阵的字符窗口，满屏显示 80 列×25 行字符，对应分辨率为 720×350 像素。

**CGA (Color Graphics Adapter):** 彩色图形与字符显示适配器，可以兼容字符和图形两种显示方式。字符分辨率为 40×25 或 80×25，图形分辨率为 320×200 或 640×200，前者可选用 4 种颜色，后者选用两种颜色。

EGA（Enhanced Graphics Adapter）：增强图形适配器，显示标准完全兼容 CGA 和 MDA 等各种显示方式，在图形方式下分辨率为  $640 \times 350$ ，16 种颜色。

VGA（Video Graphics Array）：视频图形阵列适配器，它完全兼容 EGA 的显示标准，字符窗口为  $9 \times 16$  点阵，图形方式下分辨率为  $640 \times 480$ ，16 种颜色，或  $320 \times 200$ ，256 种颜色。

SVGA（Super VGA）：除兼容 MDA、CGA、EGA、VGA 的显示方式外，还支持  $1280 \times 1024$  像素，每像素点有  $2^{24}$  种颜色，刷新频率可达 75MHz。

MDA、CGA、EGA、VGA 的显示方式如表 10-3 所示。

表 10-3 MDA、CGA、EGA、VGA 的显示方式

方式	类型	颜色数	分辨率	字符矩阵	VRAM	适配器
0, 1	文本	16	$320 \times 200$	$8 \times 8$	B8000H	CGA 兼容
2, 3	文本	16	$640 \times 200$	$8 \times 8$	B8000H	CGA 兼容
4, 5	图形	4	$320 \times 200$	$8 \times 8$	B8000H	CGA 兼容
6	图形	2	$640 \times 200$	$8 \times 8$	B8000H	CGA 兼容
7	文本		$720 \times 350$	$9 \times 14$	B0000H	MDA 兼容
0DH	图形	16	$320 \times 200$	$8 \times 8$	A0000H	EGA 兼容
0EH	图形	16	$640 \times 200$	$8 \times 8$	A0000H	EGA 兼容
0FH	图形		$640 \times 350$	$8 \times 14$	A0000H	EGA 兼容
10H	图形	16	$640 \times 350$	$8 \times 14$	A0000H	EGA 兼容
11H	图形	2	$640 \times 480$	$8 \times 16$	A0000H	VGA
12H	图形	16	$640 \times 480$	$8 \times 16$	A0000H	VGA
13H	图形	256	$320 \times 200$	$8 \times 8$	A0000H	VGA

液晶显示器使用两种视频接口标准，一种是 VGA 模拟接口，另一种是 DVI 数字接口。多数液晶显示器使用 VGA 视频标准，这是因为 PC 通常采用 VGA 标准接口。由于液晶显示器本身只能处理数字信息，因此如果使用 VGA 接口，就必须先把输入的模拟信号转换成数字信号，然后再进行处理，这一过程会损失不少信息，导致图像质量下降。DVI（Digital Visual Interface）是数字视频接口，可以把计算机产生的数字信号直接输出给显示器，不需要进行任何转换。

### 1. CGA适配器

CGA 采用 Motorola 公司的 MC 6845 芯片作为 CRT 控制器，有 7 种显示方式。

MC 6845 的内部寄存器总计为 19 个，其中 1 个用做内部地址寄存器，余下的 18 个寄存器 R0~R17 统称为数据寄存器。当 CPU 要同 MC 6845 内部的某个数据寄存器交换数据时，必须先把这个数据寄存器的编号（寄存器的地址）送到它的内部地址寄存器中。系统是指向 MC 6845 的内部地址寄存器还是数据寄存器，由 MC 6845 的 RS 端是低电平还是高电平决定。这样，MC 6845 仅需占用系统 CPU 的 2 个口地址，分别指向它的内部地址寄存器和数据寄存器。

MC 6845 的功能是，一方面要按照一定的频率产生水平同步信号、垂直同步信号和显示允许信号，将这些信号送到视频信号处理电路；另一方面要按着字符时钟周期读取显示存储器中的字符码，并且通过字符发生器转换成点阵码送到视频信号合成器。为此，要产生访问显示存储器的地址以及字符发生器的行选择地址。

CGA 提供 3 种端口。① 直接驱动端口，它通过一个 9 针 D 型插座可以直接和彩色显示器相连，用来驱动彩色显示器。这时，显示器控制板输出红、绿、蓝三色信号和亮度信号，还有水平同步信号和垂直同步信号。② 合成视频信号端口，此端口通过一个 2 针插头可以连接家用电视接收机的视频放大级。这种情况下，显示器控制板输出一个复合的视频信号，它包含了水平同步分量、垂直同步分量、颜色分量和亮度分量。③ 专用于驱动家用电视机的端口，这时，显示器接口板通过一个 4 针插头连接到一个外加的射频调制器。这样，显示器控制板输出的复合视频信号可以被调制到一个未占用的电视频道，然后，调制器将调制好的信号送到电视接收机的天线入口，由电视机直接接收。

CGA 适配器中主要有显示存储器 VRAM、CRT 控制器、字符发生器 ROM、彩色编码器等部门件。

VRAM 的地址为 0B8000H~0BC000H，它提供 16KB 的存储空间。VRAM 既可以被 CPU 访问，又可以被显示器控制部件 MC 6845 访问。在高分辨率字符方式下，规定 CPU 只能在回扫期间访问 VRAM。在其他方式下，则规定 CPU 和 MC 6845 对 VRAM 的访问采用分时方式。

CRT 控制器一方面提供光栅扫描所需要的水平同步信号和垂直同步信号，另一方面提供访问 VRAM 的地址信息。由于 MC 6845 有很强的可编程能力，所以，通过编程能很灵活地适应各种工作方式，以满足一些特殊的要求。

字符发生器 ROM 由 8KB ROM 组成，内部存放了 3 种不同的字模点阵码。一种是 7×7 的双点字模点阵码，另一种是 5×7 的单点字模点阵码，还有一种是 7×9 的普通字模点阵码。对于彩色显示器来说，由于分辨率的原因，只能使用前两种字模。一般系统中，常使用 7×7 的双点字模。

彩色编码器是一个重要部件，它的功能是根据字符各点阵代码、图形的点阵码和配色器输出产生相应的红、绿、蓝和亮度信号，并输出到彩色显示器。

#### (1) 字符显示

在字符显示方式中，屏幕上的每个显示字符在显示缓冲区中都用 2 字节来表示，其中，1 字节是字符的 ASCII 码，另 1 字节是字符属性码。ASCII 码一定存放在偶地址单元中，属性码一定存放在奇地址单元中。

因为 CGA 拥有 16KB 的存储容量，所以对 40 列×25 行的字符显示，可以同时存储 8 页的字符；而对 80 列×25 行的字符显示，则只能存储 4 页字符。每一页都可以由任何偶数地址开始，对 40 列×25 行的字符显示，用到其后的 2KB 容量，而对 80 列×25 行的字符显示，则用到其后的 4KB 容量。

#### (2) 图形显示

CGA 在图形显示时，低分辨率显示为 160×100，每点可以选择 16 种彩色中的一种。ROM BIOS 不支持这种方式，需要时要由用户自行开发。CGA 通常使用中分辨率显示和高分辨率显示。中分辨率的显示是 320×200，每个点可以选 4 种彩色中的一种。高分辨率的显示是 640×200，每个点可以选 2 种彩色中的一种。

中分辨率为 320×200，即 64 000 个点，而适配器中的 VRAM 只有 16KB。因此，

VRAM 中的每一字节对应 4 个点的代码，即每个点对应 2 位二进制信息。由于 2 位有 4 种组合，故中分辨率每一幅特定的图形，只能有 4 种彩色。因为每一字节可以显示 4 个点，所以需要 80 字节显示 1 行的 320 个点。

在高分辨率图形显示中，要显示  $640 \times 200$  个黑白点的（实际上是单色）图形。对于这种显示，每一行需要 80 字节显示 1 行的 640 个点。这时，每一字节代表 8 个点，每一位代表 1 个点。字节中的  $D_7$  位对应第 0 个像素， $D_6$  位对应第 1 个像素， $D_0$  位对应第 7 个像素。0 表示背景颜色（黑色），1 表示某一种颜色（不一定是白色），它是由彩色选择寄存器确定的某一种颜色。

## 2. EGA适配器

EGA 保留了 CGA 的所有功能，支持 CGA 的所有显示方式，同时增加了 07H、0DH、0EH、0FH、10H 等显示方式。

EGA 的 VRAM 为 64KB，最多可扩展到 256KB。在图形方式下，EGA 的 VRAM 地址因显示方式的不同而有所变化。当显示方式为 0DH~10H 时，VRAM 地址为 A0000H，这时不分奇数区和偶数区，整个 VRAM 连续地映射屏幕像素。此时的 VRAM 可以看做由 4 个位平面组成，这 4 个位平面使用相同的地址空间，有专用逻辑来识别不同的位平面。

EGA 有 64 个 6 位的颜色寄存器，用于预先设定颜色。颜色寄存器的位 5 至位 0 分别代表输出的颜色分量，其中 r、g、b 是对应的 R、G、B 的加亮标志。颜色寄存器的内容是固定的，数值是 0~63，不能改变，代表不同的颜色，有 64 种颜色。

EGA 还有一个调色板，是 16 个 6 位的颜色索引寄存器，用于选择当前可用的颜色。颜色索引寄存器中的数值用做选择颜色寄存器的指针，16 个颜色索引寄存器的数值组合起来，形成屏幕能同时显示的 16 种颜色。

EGA 的 10H 显示方式的工作原理如图 10-24 所示。

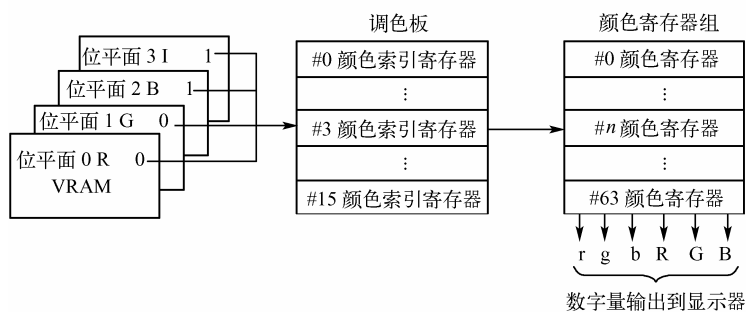


图 10-24 EGA 的 10H 显示方式工作原理

EGA 的 VRAM 采用位平面结构，将 VRAM 分成大小相同的 4 个位平面，4 个位平面的地址是重叠的。CPU 访问 VRAM 的某一个单元时，对卡上的图形控制器进行设置，实现同时读/写其中一个或几个位平面。4 个位平面分别代表 RGBI（I 为亮度）每个位平面中的一位二进制数代表一个像素点，1 字节包含 8 个像素，4 个位平面同一地址字节中相同位置的一位二进制数组合起来，便可代表该像素的颜色信息。4 个位平面的 4 位二进制数形成 0~15 的数值信息，对应地选择调色板中的一个颜色索引寄存器，而该颜色索引寄存器中的

数值，又去对应地选择颜色寄存器组中的一个颜色寄存器，该颜色寄存器中的数值，就是最后选择的颜色组合，直接输出送到显示器。

### 3. VGA适配器

VGA 兼容了 EGA 的全部功能，支持 EGA 的所有显示方式，同时增加了 11H、12H、13H 等显示方式。

在图形方式下，VGA 的 VRAM 与 EGA 相同。

EGA 的颜色寄存器的内容是固定的，而 VGA 的颜色寄存器采用 RAM，其内容是可以改变的，同时，寄存器的个数增加到 256 个，宽度增加到 18 位，RGB 每种颜色为 6 位，总颜色数为  $2^{18}$ ，即 256K 种颜色。颜色寄存器中的数值通过 3 个 6 位的 DAC 分别变成 64 级 RGB 模拟信号。

VGA 的调色板与 EGA 基本一样，但为了能寻址到 256 个颜色寄存器，增加一个颜色选择寄存器，由调色板和颜色选择寄存器的内容共同组成一个 8 位的地址。调色板和颜色选择寄存器有两种组合方式，一种是由颜色选择寄存器的低 4 位和调色板的低 4 位组成的 8 位地址，另一种是由颜色选择寄存器的低 2 位和调色板的低 6 位组成的 8 位地址。

典型的 VGA 显示方式是 12H，其工作原理如图 10-25 所示，从图中可以明显看出 VGA 与 EGA 的区别。

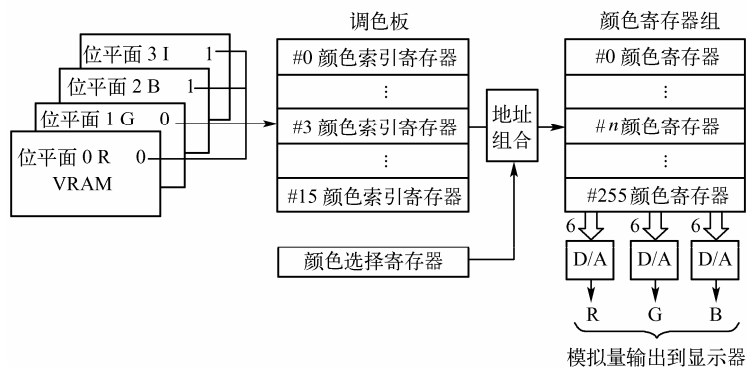


图 10-25 VGA 的 12H 显示方式工作原理

13H 是 VGA 的一种特殊的图形显示方式，图 10-26 是其工作原理示意图。该显示方式下，VRAM 不再被划分为位平面结构，也不再使用调色板，VRAM 中每一字节对应屏幕上的一个像素，存储地址按显示行顺序线性排列，字节的内容直接作为颜色寄存器的地址指针，表示该像素的颜色号，而真正的颜色由对应的颜色寄存器中的 RGB 数值确定。

VGA 的操作比较复杂，系统 BIOS 没有实现全部功能。VGA 专门配备了视频 BIOS ROM，机器启动时，由硬件装置将控制权转换到 VGA 的 ROM 上，置换原系统 BIOS 的视频中断向量，由 VGA

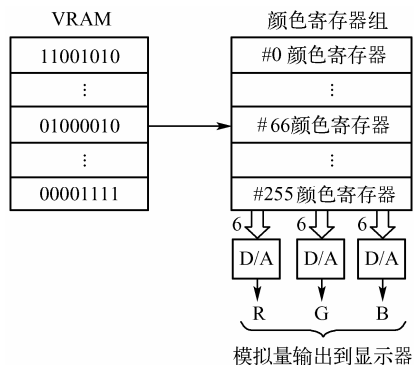


图 10-26 VGA 的 13H 显示方式工作原理



上的 ROM 程序完成相应的显示操作。

### 10.5.6 对显示器的编程

对显示器的编程与其他通用硬件编程一样，可以直接对硬件编程，也可以直接调用 ROM BIOS 中的显示 I/O 功能程序或有关 DOS 功能调用，前者，必须弄清硬件细节才能有的放矢地编程，而后者只须了解具体调用方法。可通过 DOS 功能调用中“INT 21H”子功能，即选择 AH=1、2、9、0AH 等可方便地实现字符和字符串的输出和显示。对于常规图形显示方式，通过调用 BIOS 功能就能实现，而且非常简便。下面仅介绍采用 BIOS 功能调用的方法实现对显示器的编程。

BIOS 中具有驱动显示适配器功能的程序，它包括 16 个功能模块，只要给 AH 装入指定的模块号，再执行一条“INT 10H”指令，即可调用它的一个功能模块。

#### 1. BIOS 设置显示方式

显示方式分为文本方式和图形方式，文本方式是默认方式。在图形方式下，可通过读写屏幕上各个像素点的映像，显示出图形。BIOS 提供了设置文本和图形显示方式的功能，程序只要给出调用参数，使用 BIOS 的“INT 10H”即可建立某种显示方式。

利用 BIOS “INT 10H”的功能 00 可为当前的执行程序初始化显示方式，或在文本方式和图形方式之间切换。

入口参数：AH=00H

AL=显示方式号

【例 10-1】 设置 VGA 图形方式。

```
MOV AH, 00H
MOV AL, 12H
INT 10H
```

【例 10-2】 设置 VGA 文本方式。

```
MOV AH, 00H      ; 0 号功能为显示方式选择
MOV AL, 03H      ; 方式 3，彩色字符
INT 10H          ; 调用 BIOS
```

#### 2. 设置光标位置

入口参数：AH=02H

BH=光标新位置的页号(图形方式置 0)

DH=光标新位置的字符行号

DL=字符列号

【例 10-3】 在显示页 0 置光标(2, 15)，即第 0 页第 15 行第 2 列字符处。

```
MOV AH, 2
MOV BH, 0      ; 0 页
MOV DL, 2      ; 列号
```

MOV DH, 15 ; 行号  
INT 10H ; 调用 BIOS

### 3. 设置显示的页

利用 BIOS “INT 10H” 的功能 5H，可以很方便地设置实际显示的页。因 VRAM 中可有几页显示的信息，但一次只能显示 1 页，本调用可选择显示哪一页。

入口参数：AH=05H

AL=要显示的页号

页号数与显示方式及 VRAM 容量有关，  
如表 10-4 所示。

### 4. 显示字符串

利用 BIOS “INT 10H” 的功能 13H，可以很方便地在屏幕上显示字符串。本功能调用将存储器内的字符串及属性显示到屏幕。

入口参数：AH=13H

BH=显示的页号

CX=字符计数（串的长度）

DH=串开始的行号

DL=串开始的列号

ES:BP=字符串所在的段和偏移地址

AL=方式

AL=0 BL 为所有字符的属性，光标不移动

AL=1 BL 为所有字符的属性，光标移动

AL=2 ASCII 码和属性的字符串，光标不移动

AL=2 ASCII 码和属性的字符串，光标移动

表 10-4 页号数与显示方式及 VRAM 容量对应表

显示方式	可用页号		
	64KB	128KB	256KB
0, 1	0~7	0~7	0~7
2, 3	0~3	0~7	0~7
4, 5, 6	0	0	0
7	0~3	0~7	0~7
0DH	0~1	0~3	0~7
0EH	0	0~1	0~3
0FH	0	0~1	0~1
10H	0	0	0~1
11H			0（仅 VGA）
12H			0（仅 VGA）
13H			0（仅 VGA）

### 5. 读像素

在 EGA/VGA 图形方式下，利用 BIOS “INT 10H” 的功能 0DH，可以很方便地读入任意页的一个像素。调用时，程序员提供颜色、页号、行号和列号。

入口参数：AH=0DH

AL=颜色号（像素颜色，取决于显示方式）

BH=显示页号

CX=像素的列号（0~319 或 0~639）

DX=像素的行号（0~199 或 0~349，0~479）

返回参数：AL=颜色值

### 6. 写像素

在 EGA/VGA 图形方式下，利用 BIOS “INT 10H” 的功能 0CH，可以很方便地写一个像素点到 VRAM。调用时，程序员提供颜色、页号、行号和列号。

利用该功能，可以将一个像素点写到像素位置。在合适的显示方式下，还可以指定

颜色。

入口参数：AH=0CH

AL=颜色号（像素颜色，取决于显示方式）

BH=显示页号

CX=像素的列号（0~319 或 0~639）

DX=像素的行号（0~199 或 0~349, 0~479）

返回参数：无

当设置 AL 的 7 位为 1 时，会引起新的像素值与当前像素值进行异或操作。

**【例 10-4】** 用 BIOS 的功能调用“INT 10H”，AH=0H 功能来设置屏幕为 640×480 彩色图形方式（显示方式 AL=12H），并从屏幕第 10 行画一条由 16 种颜色像素点组成的线（用功能调用 0CH）。程序结束时，显示器仍在 12H 下。

程序如下：

```
DATA    SEGMENT
YS      DB  16                      ;颜色初始值为 16
DATA    ENDS
CODE    SEGMENT
        ASSUME CS:CODE,DS:DATA
MAIN    PROC
START:  MOV     AX, DATA
        MOV     DA, AX
        MOV     AH, 0                ; 选择功能调用 0, 设置显示方式
        MOV     AL, 12H              ; 设置显示方式 12H
        INT     10H
        MOV     CX, 639              ; 共 640 列
LP1:    MOV     AH, 0CH              ; 写像素点功能调用
        DEC     YS                   ; 颜色号减 1
        MOV     AL, YS
        JNZ     SKIP1               ; 颜色号不为 0 转 SKIP1
        MOV     YS, 16              ; 恢复 YS 变量为 16
SKIP1:  MOV     BH, 0                ; 选择 0 页
        MOV     DX, 10              ; 在第 10 行显示
        INT     10H
        LOOP    LP1                 ; 列号减 1, 继续循环显示
        DEC     YS                   ; 颜色号减 1
        MOV     AL, YS
        INT     10H
        MOV     AH, 4CH
        INT     21H
```

```

MAIN    ENDP
CODE    ENDS
END      START

```

【例 10-5】 下列程序将屏幕的第 10 行复制到第 20 行，该程序应在显示器设置成高分辨率图形方式下运行。判断当前显示方式是否为高分辨率，如果不是，则转移到 BAD\_MODE，并显示“The mode must be set high resolution”。

程序如下：

```

DATA    SEGMENT
MSG      DB  'The mode must be set high resolution'
N        EQU    $-MSG          ; 显示字符个数
DATA    ENDS
CODE    SEGMENT
        ASSUME CS:CODE,DS:DATA,ES:DATA
MAIN    PAR
START:  MOV     AX, DATA
        MOV     DA, AX
        MOV     AH, 0           ; 选择功能调用 0，设置显示方式
        MOV     AL, 10H        ; 设置显示方式 10H
        INT     10H
        MOV     AH, 0FH        ; 读显示方式
        INT     10H
        CMP     AL, 0EH        ; 若小于 0EH 显示方式则转
        JL      BAD_MODE
        CMP     AL, 13H        ; 若是 256 色的 13H 显示方式则转
        JZ      BAD_MODE
        MOV     CX, 639        ; 开始列号
LP1:    MOV     AH, 0DH        ; 读像素点功能调用
        MOV     BH, 0          ; 0 页
        MOV     DX, 10         ; 10 行
        INT     10H
        MOV     AH, 0CH        ; 写像素点功能调用
        MOV     DX, 20         ; 20 行
        INT     10H
        DEC     CX
        CMP     CX, 0FFFFH
        JNZ     LP1
        MOV     AH, 4CH
        INT     21H

```

```
BAD_MODE:  MOV     AX, DATA
            MOV     ES, AX
            MOV     AH, 3          ; 读光标位置功能调用
            MOV     BH, 0          ; 0 页
            INT     10H           ; DX 中有光标位置
            MOV     DL, 0          ; 选 0 列
            MOV     AX, 1300H      ; 写字串
            MOV     BH, 0          ; 0 页
            MOV     BL, 3          ; 颜色号为 3
            LEA     BP, MSG        ; 字节的偏移地址
            MOV     CX, N          ; 字符串长度
            INT     10H
            MOV     AH, 4CH
            INT     21H
MAIN       ENDP
CODE      ENDS
END        START
```

## 10.6 鼠标器及其接口

鼠标器的英文名称为“Mouse”，也就是“老鼠”的意思，其外型轻巧（如图 10-27 所示），应用自如。它的运用，使计算机用户不必再记忆繁多的命令，只须移动鼠标，将光标移至有关命令的位置处，再单击鼠标，即可执行相应的操作，大大提高了计算机的使用效率。目前鼠标因其性能好，价格低，操作简便，已成为计算机的主要外设之一，尤其对现在的 Windows 操作系统而言，鼠标已经是不可或缺的基本设备了。

除了鼠标的外型，有些鼠标的底部会有一个切换开关，开关的两侧会印有 MS 及 PC 字样，这两个字样有什么意义呢？我们知道，目前全世界最大的软件公司当属 IBM（PC-DOS、OS 操作系统的厂商）及 Microsoft 公司，然而这两家公司对鼠标按键的定义却各有不同的看法。Microsoft 公司认为鼠标只要两个按键就可以应付所有的操作；而 IBM 公司（PC）则认为：起码要 3 个按键才可以做到这一点。



图 10-27 双飞燕 3D 无线鼠标

定义的不同，给鼠标的制造厂商带来了麻烦，不过鼠标制造厂商也有自己的办法，他们将两种不同定义的鼠标制作成“二合一”的鼠标，以切换开关的方法，来兼容不同的操作系统。可见，如果使用的操作系统是 Windows 95 或 Windows 98，就必须将开关切换到“MS”位置；而如果使用的操作系统是 IBM 的 PC-DOS、OS/2 等，则应该将开关切换到“PC”位置。

鼠标按照工作原理分类，可以分为“机械鼠标”和“光电鼠标”两类。

机械鼠标最常见，也是我们用得最多的一种鼠标，其工作原理是由底部的一颗塑胶球的滚动而传动两个垂直与水平的圆轴，这两个轴的另一端则分别安装了一个光控转盘，转盘上有许多的细缝，当光控转盘转动时，光控开关就会因为光控转盘的细缝转动而传出一开一关的信号给鼠标的电路板，鼠标再将接收到的信号传给微机。这种装置就叫做机械鼠标，其结构原理如图 10-28 所示。

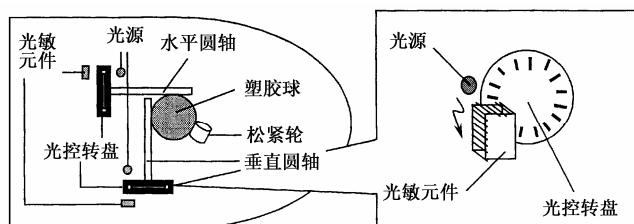


图 10-28 机械鼠标器结构原理

光电鼠标与机械鼠标从外观上讲没有大的区分，但工作原理却大不一样。它是由一个发光二极管和一个感光接收器所组成的，由于感光接收器需要光源反射，因此，这种鼠标还需要一个特殊的鼠标垫（Mouse PAD）来达到反射光源的目的，这种鼠标垫的表层一般用金属薄膜加网状条纹制成。光电鼠标器的结构原理如图 10-29 所示。

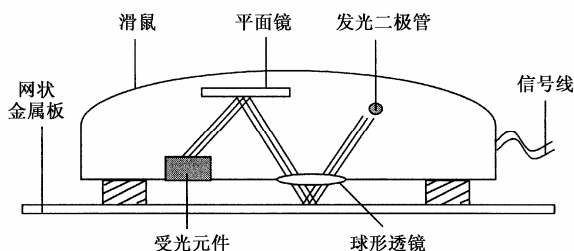


图 10-29 光电鼠标器的结构原理

在 Windows 95 或 Windows 98 操作系统下，90%以上的功能，都使用鼠标左键来操作。通常执行功能表的指令、选取文字或图案、按下对话框中的按钮或选项，都要单击鼠标左键；而鼠标右键通常用在显示快速选单。当单击鼠标右键时，出现的选单里会有一些常用的指令，直接在指令上单击鼠标左键，就可以快速执行这个功能。如果使用的是三键鼠标，通常中键没有什么功能，不过有些鼠标可以由使用者自行设定，让鼠标中键来取代某些功能，例如取代双击鼠标左键等。此外，有些特殊设计的鼠标中间会有滚轮，通常用来快速滚动画面，如浏览网页等。

## 10.7 其他外部设备

### 10.7.1 扫描仪

扫描仪是图像信号输入设备。它对原稿进行光学扫描，然后将光学图像传送到光电转

换器中变为模拟信号，并将模拟信号变换成为数字信号，最后通过计算机接口送至计算机中，其扫描示意图如图 10-30 所示。

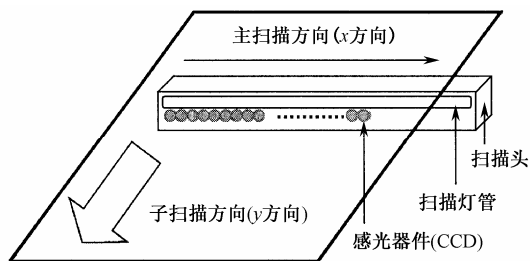


图 10-30 台式平板扫描仪扫描示意图

扫描仪扫描图像的步骤：

（1）首先将欲扫描的原稿正面朝下铺在扫描仪的玻璃板上，原稿可以是文字稿件或者图纸照片。

（2）启动扫描仪驱动程序后，安装在扫描仪内部的可移动光源开始扫描原稿。为了均匀照亮稿件，扫描仪光源为长条形的，并沿  $y$  方向扫过整个原稿。

（3）照射到原稿上的光线经反射后穿过一个很窄的缝隙，形成沿  $x$  方向的光带，又经过一组反光镜，由光学透镜聚焦并进入分光镜，经过棱镜和红绿蓝三色滤色镜得到的 RGB 三条彩色光带分别照到各自的 CCD 上，CCD 将 RGB 光带转变为模拟电信号，此信号又被 ADC 转换为数字电信号，扫描方式如图 10-31 所示。至此，反映原稿图像的光信号转换为计算机能够接收的二进制数字信号，最后通过串行或者并行等接口送至计算机。扫描仪每扫一行就得到原稿  $x$  方向一行的图像信息，随着沿  $y$  方向的移动，在计算机内部逐步形成原稿的全图。

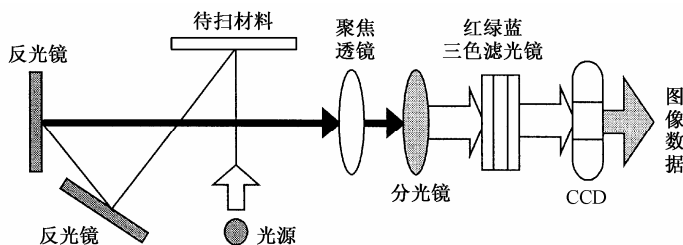


图 10-31 图像扫描方式

在扫描仪获取图像的过程中，有以下两个元件起到关键作用，其性能直接影响扫描仪的整体性能指标。

一个是 CCD，它将光信号转换成为电信号。CCD 是 Charge Couple Device 的缩写，称为电荷耦合器件，它是利用微电子技术制成的表面光电器件，可以实现光电转换功能。CCD 在摄像机、数码相机和扫描仪中应用广泛。摄像机中使用的是点阵 CCD，即包括  $x$ 、 $y$  两个方向，用于摄取平面图像；扫描仪中使用的是线阵 CCD，它只有  $x$  一个方向， $y$  方向扫描由扫描仪的机械装置来完成。CCD 芯片上有许多光敏单元，它们可以将不同的光线转换成不同的电荷，从而形成对应原稿光图像的电荷图像。如果想增加图像的分辨率，就必须增加 CCD 上的光敏单元数量。实际上，CCD 的性能决定了扫描仪的  $x$  方向的光学分辨率。

另一个是 ADC，它将模拟信号变为数字信号。从 CCD 获取的电信号是对应于图像明暗的模拟信号，也就是说图像由暗到亮的变化可以用从低到高的不同电平来表示，它们是连续

变化的,即所谓的模拟量。ADC 的工作是将模拟量数字化。例如,将  $0\sim 1\text{V}$  的线性电压变化表示为  $0\sim 9$  的 10 个等级的方法是:  $0\sim 0.1\text{V}$  的所有电压都转换为数字 0,  $0.1\sim 0.2\text{V}$  的所有电压都转换为数字 1…… $0.9\sim 1.0\text{V}$  的所有电压都转换为数字 9。实际上,ADC 能够表示的范围远远大于 10。通常是,  $2^8=256$ 、 $2^{10}=1024$ 、 $2^{12}=4096$ ,等等。如果扫描仪说明书上标明的灰度等级是 10bit,则说明这个扫描仪能够将图像分成 1024 个灰度等级;如果标明色彩深度为 30bit,则说明红、绿、蓝各个通道都有 1024 个等级。显然,该等级数越高,表现的彩色越丰富。

## 10.7.2 绘图仪

绘图仪是输出高质量图形的设备。打印机虽然也能输出图形,但是质量、速度比不上绘图仪。绘图仪的种类很多,基本上可分为滚筒式和平台式两类,它们的组成部分有驱动电机、机械传动部分、控制电路、绘图台笔架、插补器等。

### 1. 滚筒式绘图仪

滚筒式绘图仪工作原理如图 10-32 所示。送纸机构和笔架机构直接影响绘图质量,是绘图仪的主要机构。

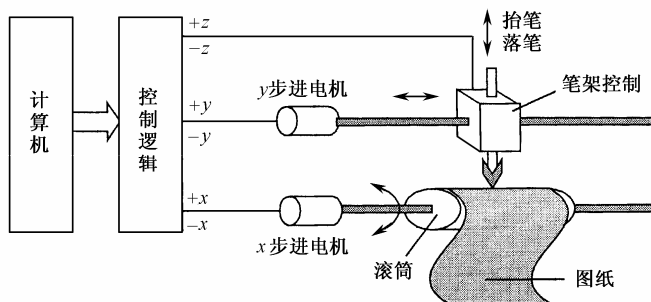


图 10-32 滚筒式绘图仪工作原理

送纸机构由送纸电机和收纸电机、滚筒等组成。图纸在绘图过程中正向和反向移动。卷筒绘图纸分别绕在送纸和收纸电机带动的转轴上,中间经过带有链齿的滚筒,滚筒上齿的间距与卷筒绘图纸两边孔的孔距应一致。绘图过程中图纸位置的检测由电气反馈系统控制,以便及时发出信号控制送纸和收纸电机。

绘图笔的质量会直接影响绘图质量。目前采用的绘图笔有圆珠笔、钢笔等。绘图笔的抬笔和落笔动作由绘图指令控制电磁机构来实现。在高速绘图时,抬笔和落笔动作十分频繁,为了避免绘图笔高速冲击纸面产生跳跃而使图形断线,在绘图笔上装有动圈机构以减轻笔座的负担,并用控制流入线圈电流大小的办法,使高速下降的绘图笔在接触纸面的瞬间产生制动作用,从而使绘图笔与纸面平稳地接触。

图 10-32 中,当  $x$  方向步进电机通过传动机构驱动滚筒转动时,就带动图纸转动(可正、可反),即可在  $x$  方向运动。 $y$  方向步进电机驱动笔架,可在  $y$  方向移动。 $z$  方向步进电机控制抬笔和落笔。计算机送来的绘图信息通过控制装置驱动  $x$ 、 $y$ 、 $z$  方向的运动。滚筒式绘图仪结构紧凑、占地面积小、绘图幅面大,但图纸要有链孔。



## 2. 平台式绘图仪

平台式绘图仪工作原理如图 10-33 所示。绘图仪平台台面一般由硬质橡皮构成，其上有  $x$  方向和  $y$  方向两组导轨。 $x$  方向导轨位于平台上， $y$  方向导轨位于横梁上，横梁可沿  $x$  导轨滑动，从而产生笔架的  $x$  方向运动。笔架可沿着  $y$  方向导轨滑动。把这两个方向的运动组合起来，就能使绘图笔实现绘图所需要的运动。此外还有抬笔落笔机构。将图纸固定在平台上有 3 种方式：① 真空吸附式，在台面上开一些小孔，真空泵在台面下抽气把纸吸附在平台上。② 静电吸附，在台面上加高压电场，将纸吸在平台上。③ 磁条压紧，用铁磁材料做台面，用磁条将图纸压紧。

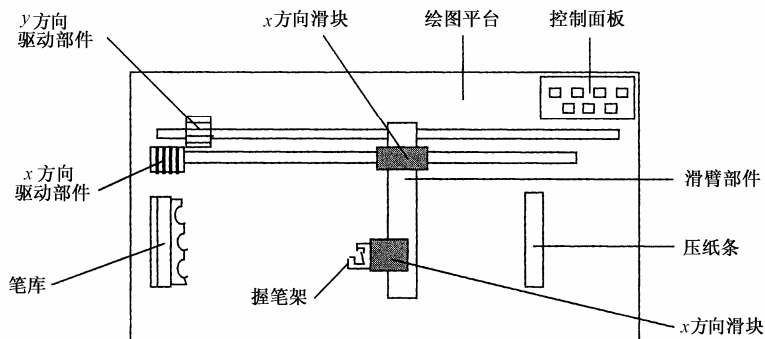


图 10-33 平台式绘图仪工作原理图

平台式绘图仪由控制面板、电源、接口、控制电路、驱动电路、机械传动等部分组成。

## 习题

1. 打印机分为哪几类？点阵打印机属于哪类？它由哪几部分组成？
2. LED 显示器的工作原理是什么？何谓共阳极？何谓共阴极？
3. LED 显示器接口有哪几种方式？
4. 简述 LED 硬件译码显示的工作原理。
5. 简述 LED 软件译码动态显示接口设计中的要点。
6. 设计一个通过 8255A 芯片控制的一个两位 LED 的动态显示接口电路。
7. CRT 接口中显示存储器的内容和 CRT 显示画面是如何对应的？显示存储器的内容与字符发生器有什么关系？
8. 为什么键盘输入时，需考虑去抖动问题？设计一个去抖动电路。
9. 从键盘提供编码形式来看，键盘分为哪几类？各有什么优缺点？
10. 什么是行扫描法和线反转法？其实现过程有哪些区别？
11. 简述打印机接口中的主要信号。
12. 简述在查询方式下，打印机的工作原理。
13. 现行 PC 的打印机与主机间最常用的接口是（ ）。

- A. IEEE 488 接口                      B. Centronics 接口  
C. RS-232C 接口                      D. IDE 接口

14. 平台式绘图仪是怎样工作的?

15. 微机常见的输入设备有\_\_\_\_\_, 常见的输出设备有\_\_\_\_\_和\_\_\_\_\_, 绘图仪是\_\_\_\_\_设备。

16. 某显示器的分辨率为  $1024 \times 768$ , 则全屏幕像素个数为\_\_\_\_\_。

17. 简述键盘扫描的主要任务。

18. 设有 14 个按键组成键盘阵列, 识别这 14 个按键至少需要有 ( ) 根口线。

- A. 6                      B. 7                      C. 8                      D. 14

19. INT 09H 的作用是 ( )。

- A. 将扫描码解释成为系统信号和缓冲区数据  
B. 将扫描码送到 CRT 显示  
C. 将 AL 中的数据送打印机打印  
D. 将显示缓冲区中的字符按当前光标的位置在 CRT 屏幕上显示

# 第 11 章

## MCS-51 单片机

### 📖 教学目的和要求

要求掌握 MCS-51 系列单片机的结构、特性、指令、寻址方式、伪指令、简单的汇编语言程序设计，熟悉中断定时功能，了解实时控制 I/O 接口的基本方法，为更好地应用单片机打好基础。

本章主要以 MCS-51 系列的 8051 为典型例子, 介绍单片机的结构、性能、各个引脚的功能、存储器配置及工作原理等内容。

## 11.1 MCS-51 单片机的组成

MCS-51 单片机典型产品有 8031、8051、8751。8051 内部有 4KB ROM, 8751 片内有 4KB EPROM, 8031 无片内 ROM。除此之外, 三者的内部结构及引脚完全相同。

8051 单片机结构框图见图 11-1。8051 单片机内部结构中各个部件的功能简介如下:

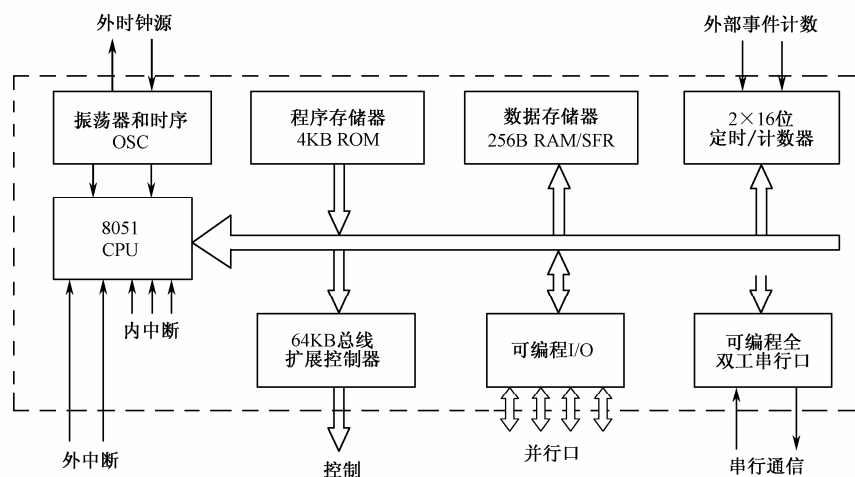


图 11-1 8051 单片机结构框图

① 中央处理器。中央处理器是单片机的核心, 完成运算和控制功能。MCS-51 的 CPU 能处理 8 位二进制数。

② 内部数据存储器 RAM。共有 256 个 RAM 单元, 其中后 128 个单元作为特殊功能寄存器; 前 128 个单元用于存放可读/写的数。

③ 内部程序存储器 ROM。共有 4KB 掩膜 ROM, 用于存放程序、原始数据或表格。

④ 定时/计数器。有 2 个 16 位的定时/计数器, 以实现定时或计数功能, 并以其定时或计数结果对计算机进行控制。

⑤ 并行 I/O 口。MCS-51 共有 4 个 8 位的 I/O 口 ( $P_0$ 、 $P_1$ 、 $P_2$ 、 $P_3$ ), 以实现数据的并行输入/输出。

⑥ 串行口。MCS-51 单片机有一个全双工的串行口, 用以实现单片机和其他设备之间的串行数据传送。

⑦ 中断控制系统。共有 5 个中断源, 即外中断 2 个, 定时/计数中断 2 个, 串行中断 1 个。全部中断分为高级和低级两个优先级别。

⑧ 时钟电路。MCS-51 单片机芯片的内部有时钟电路, 但石英晶体和微调电容需外接。时钟电路为单片机产生时钟脉冲序列。系统允许的最高晶振频率为 12MHz。

## 11.2 MCS-51 单片机的芯片引脚

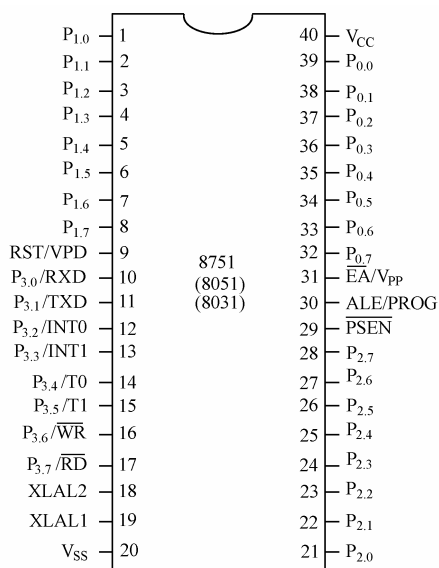


图 11-2 MCS-51 单片机芯片引脚图

MCS-51 单片机是标准的 40 引脚双列直插式集成电路芯片，引脚排列见图 11-2。

### 1. 信号引脚介绍

#### (1) I/O 端口引脚

- $P_{0.0} \sim P_{0.7}$ : P0 口 8 位双向口线。
- $P_{1.0} \sim P_{1.7}$ : P1 口 8 位双向口线。
- $P_{2.0} \sim P_{2.7}$ : P2 口 8 位双向口线。
- $P_{3.0} \sim P_{3.7}$ : P3 口 8 位双向口线。

#### (2) 控制信号引脚

- **ALE**: 地址锁存控制信号。在系统扩展时，ALE 把 P0 口输出的低 8 位地址送锁存器锁存起来，以实现低位地址和数据的隔离。此外由于 ALE 是以晶振六分之一的固定频率输出的正脉冲，因此可作为外部时钟或外部定时脉冲使用。

- **SEN**: 外部程序存储器读选通信号。在读外部 ROM 时 SEN 有效（低电平），以实现

外部 ROM 单元的读操作。

- **EA**: 访问程序存储器控制信号。当 EA 信号为低电平时，对 ROM 的读操作限定在外部程序存储器；当 EA 信号为高电平时，对 ROM 的读操作是从内部程序存储器开始的，并可延伸至外部程序存储器。

- **RST**: 复位信号。当输入的复位信号连续 2 个机器周期以上为高电平时即为有效，用以完成单片机的复位操作。

#### (3) 晶体引脚

- **XTAL1** 和 **XTAL2**: 外接晶体引脚。当使用芯片内部时钟时，此二引脚用于外接石英晶体和微调电容；当使用外部时钟时，用于接外部时钟脉冲信号。

#### (4) 电源和地线引脚

- **VSS**: 地线。
- **VCC**: +5V 电源。

### 2. 信号引脚的第二功能

由于工艺及标准化等原因，芯片的引脚数目是有限的。例如，MCS-51 系列把芯片引脚数目限定为 40 条，但单片机为实现其功能所需要的信号数目却远远超过此数，因此给一些信号引脚赋予双重功能。前述的信号定义为引脚第一功能。下面介绍某些信号引脚的第二功能。

① P3 口线的第二功能：详见表 11-1。

② EPROM 存储器程序固化所需要的信号：有内部 EPROM 的单片机芯片（例如

8751)，为写入程序需要提供专门的编程脉冲和编程电压。这些信号是由信号引脚第二功能提供的，即：编程脉冲：30 脚（ALE/PROG）；编程电压：31 脚（EA/V<sub>pp</sub>）。

表 11-1 P<sub>3</sub> 口线的第二功能

口 线	第 二 功 能	信 号 名 称
P <sub>3.0</sub>	RXD	串行数据接收
P <sub>3.1</sub>	TXD	串行数据发送
P <sub>3.2</sub>	$\overline{\text{INT0}}$	外部中断 0 申请
P <sub>3.3</sub>	$\overline{\text{INT1}}$	外部中断 1 申请
P <sub>3.4</sub>	T0	定时/计数器 0 计数输入
P <sub>3.5</sub>	T1	定时/计数器 1 计数输入
P <sub>3.6</sub>	$\overline{\text{WR}}$	外部 RAM 写选通
P <sub>3.7</sub>	$\overline{\text{RD}}$	外部 RAM 读选通

11.3 存储器配置

8051 的存储器有四个物理上相互独立的存储空间，即片内 ROM 和片外 ROM，片内 RAM 和片外 RAM。其配置如图 11-3 所示。

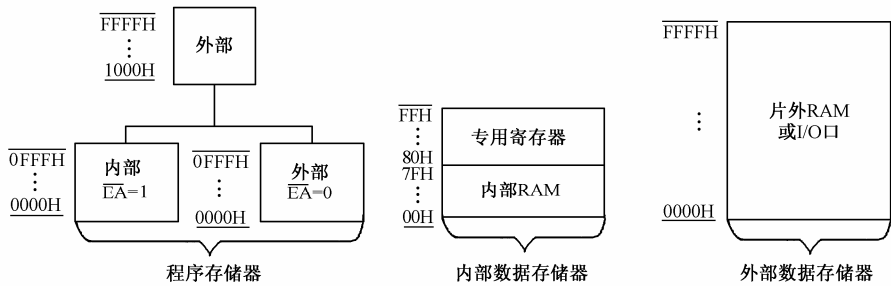


图 11-3 8051 存储器的配置图

1. 程序存储器

8051 单片机内部设置有 4KB 的 ROM，8751 单片机内部设置有 4KB 的 EPROM 作为内部程序存储器，而 8031 内部没有程序存储器，必须外接程序存储器。由于 MCS-51 单片机设置有 16 位的程序计数器，因此可以寻址 64KB 的程序存储器。程序存储器的作用在于存放编好的程序和表格常数，可通过 MOV<sub>C</sub> 指令访问程序存储器。

2. 内部数据存储器

数据存储器可分为两个地址空间：一个内部数据存储器空间和一个外部数据存储器空间。访问内部数据存储器，用 MOV 指令；访问外部数据存储器，用 MOV<sub>X</sub> 指令。8051 单片机内部设置有前 128 个单元为内部数据存储器寻址空间，后 128 个单元为特殊功能寄存器寻址空间。

(1) 内部数据存储器

内部数据存储器的前 128 地址单元的分布如图 11-4 所示。

前128单元		后128单元	
7FH	用户RAM区 (堆栈、数据缓冲)	FFH	B
⋮		F0H	
⋮		E0H	
⋮		D0H	
30H		B8H	
2FH	位寻址区 (位地址00H~7FH)	B0H	P3
⋮		A8H	
⋮		A0H	
20H		99H	
1FH		98H	
⋮	第3组寄存器区	90H	P1
18H		8DH	
17H		8CH	
⋮		8BH	
10H		8AH	
0FH	第1组寄存器区	89H	TMOD
⋮		88H	
⋮		87H	
08H		83H	
07H		82H	
⋮	第0组寄存器区	81H	SP
⋮		80H	
00H			

图 11-4 内部数据存储器

① 工作寄存器区。内部数据存储器的 00H~1FH（共 32 个单元）为 4 个寄存器工作区，每区 8 个寄存器，表示为 R0~R7。因此每次只能选择一个寄存器区工作。寄存器工作区的选择是通过状态标志寄存器 PSW 的第 3、4 位，也即 RS1、RS0 进行的，如表 11-2 所示。

表 11-2 寄存器工作区与地址分配

RS1	RS0	寄存器工作区	R0~R7 占用地址
0	0	0 区(BANK0)	00H~07H
0	1	1 区(BANK1)	08H~0FH
1	0	2 区(BANK2)	10H~17H
1	1	3 区(BANK3)	18H~1FH

② 位寻址区。内部数据存储器 20H~2FH（16 个单元）既可以按字节寻址，又可以按位由 CPU 直接寻址，进行位操作。它占用的地址为 00H~7FH（位地址），如表 11-3 所示。

表 11-3 位地址单元地址分配表

字节地址	D7~D0							
2FH	7F	7E	7D	7C	7B	7A	79	78
2EH	77	76	75	74	73	72	71	70
2DH	6F	6E	6D	6C	6B	6A	69	68
2CH	67	66	65	64	63	62	61	60
2BH	5F	5E	5D	5C	5B	5A	59	58
2AH	57	56	55	54	53	52	51	50
29H	4F	4E	4D	4C	4B	4A	49	48
28H	47	46	45	44	43	42	41	40
27H	3F	3E	3D	3C	3B	3A	39	38

续表

字节地址	D7~D0							
26H	37	36	35	34	33	32	31	30
25H	2F	2E	2D	2C	2B	2A	29	28
24H	27	26	25	24	23	22	21	20
23H	1F	1E	1D	1C	1B	1A	19	18
22H	17	16	15	14	13	12	11	10
21H	0F	0E	0D	0C	0B	0A	09	08
20H	07	06	05	04	03	02	01	00

③ 用户 RAM 区。内部 RAM 块中的 30H~7FH 构成一般缓冲存储区，可用于存放数据，也可作为堆栈存储区域。

### (2) 专用寄存器

8051 内部数据存储器的专用寄存器（SFR），其名称、地址分配见表 11-4；SFR 寄存器位的地址见表 11-5。

表 11-4 MCS-51 专用寄存器的名称、地址分配

特殊功能寄存器	功 能 名 称	地 址
B	通用寄存器	F0H
A	累加器	E0H
PSW	程序状态寄存器	D0H
IP	中断优先级控制寄存器	B8H
P3	P3 口数据寄存器	B0H
IE	中断允许控制寄存器	A8H
P2	P2 口数据寄存器	A0H
SBUF	串行口发送/接收缓冲器	99H
SCON	串行口控制寄存器	98H
P1	P1 口数据寄存器	90H
TH1	T1 计数器高 8 位	8DH
TH0	T0 计数器高 8 位	8CH
TL1	T1 计数器低 8 位	8BH
TL0	T0 计数器低 8 位	8AH
TMOD	定时/计数器方式控制寄存器	89H
TCON	定时器控制寄存器	88H
PCON	电源控制寄存器	87H
DPH	地址寄存器高 8 位	83H
DPL	地址寄存器低 8 位	82H
SP	堆栈指针寄存器	81H
P0	P0 口数据寄存器	80H



表 11-5 SFR 寄存器位的地址

(MSB)				(LSB)										
F0H									B					
	F7	F6	F5	F4	F3	F2	F1	F0						
E0H									A					
	E7	E6	E5	E4	E3	E2	E1	E0						
D0H									PSW					
	D7	D6	D5	D4	D3	D2	D1	D0						
B8H									IP					
	PS		PT1		PX1		PT0			PXD				
B0H									P3					
	—		—		—		—			—				
A8H									IE					
	B7		B6		B5		B4			B3	B2	B1	B0	
A0H									P2					
	EA		ES		ET1		EX1			ET0		EXD		
98H									SCON					
	AF		—		—		AC			AB		AA		A9
90H									P1					
	A7		A6		A5		A4			A3		A2		A1
88H									TCON					
	SM0		SM1		SM2		REN			TB8		RB8		TI
80H									P0					
	9F		9E		9D		9C			9B		9A		99
	97		96		95		94			93		92		91
	TF1		TR1		TF0		TR0			IE1		IT1		IE0
	8F		8E		8D		8C			8B		8A		89
	87		86		85		84			83		82		81

## 11.4 时钟电路及时序

### 1. 时钟电路

8051 内部有一个高增益反相放大器，用于构成振荡器，引脚 XTAL1 和 XTAL2 分别是此放大器的输入端和输出端。在 XTAL1 和 XTAL2 两端跨接晶体和陶瓷谐振器，就构成了稳定的自激振荡器，其发出的脉冲直接送入内部时钟电路，见图 11-5。外接晶振时， $C_1$  和  $C_2$  的值通常选择为 30pF 左右。

8051 也可采用外部振荡脉冲，外时钟信号由 XTAL2 端注入后直接送至内部时钟电路，见图 11-6，输入端 XTAL1 应接地。由于 XTAL2 端的逻辑电平不是 TTL 的，故建议连接一个上拉电阻。

### 2. CPU时序及有关概念

我们已知 CPU 执行指令的一系列动作都是在时序电路控制下一拍一拍地进行的，由于指令的字节数不同，那么取这些指令所需要的时间就有很大的不同，即使是字节数相同的指

令，由于执行操作有较大差别，因而不同指令的执行时间就不一定相同。为了便于说明，人们按指令的执行过程规定了 3 种周期，即时钟周期、机器周期和指令周期，下面分别予以说明。

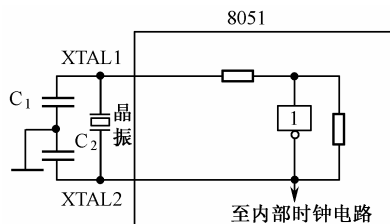


图 11-5 时钟振荡电路

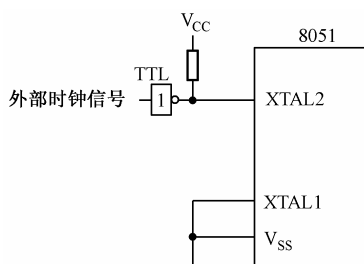


图 11-6 外部脉冲源接法

### （1）时钟周期

时钟周期也称为状态周期（S）或振荡周期，它是计算机中最基本的时间单位，在一个时钟周期内，中央处理器 CPU 仅完成一个最基本的动作。

### （2）机器周期

在计算机中，为了便于管理，常把一条指令的执行过程划分为若干个阶段，每一阶段完成一项工作。例如，取指令、存储器读、存储器写等，每一项工作称为一个基本操作。完成一个基本操作所需要的时间称为机器周期。一般情况下，一个机器周期由若干个 S 周期组成。基本定时时序关系如图 11-7 所示。

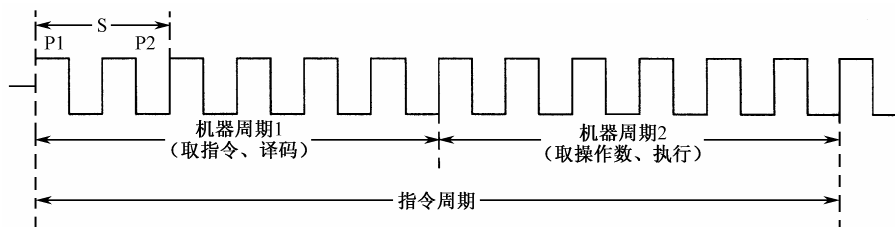


图 11-7 基本定时时序关系

### （3）指令周期

指令周期是执行一条指令所需要的时间，一般由若干个机器周期组成。指令不同，所需要的机器周期数也不同，对于一些简单的单字节指令，在取指令周期中，指令取至指令寄存器后，立即译码执行，不再需要其他的机器周期。

8051 的一个机器周期包含 6 个状态周期 S，每一个状态周期划分为两个节拍，即图 11-7 中的 P1、P2。

所以一个机器周期可依次表示为 S1P1、S1P2、S2P1、S2P2、…、S6P1、S6P2，共 12 个时钟周期。

图 11-8 列举了典型指令的取指和执行时序。我们可以通过观察 XTAL2 和 ALE 端信号，分析 CPU 取指时序。由图可知，在每个机器周期内，地址锁存控制 ALE 两次有效，第一次出现在 S1P2 和 S2P1 期间，第二次出现在 S4P2 和 S5P1 期间。

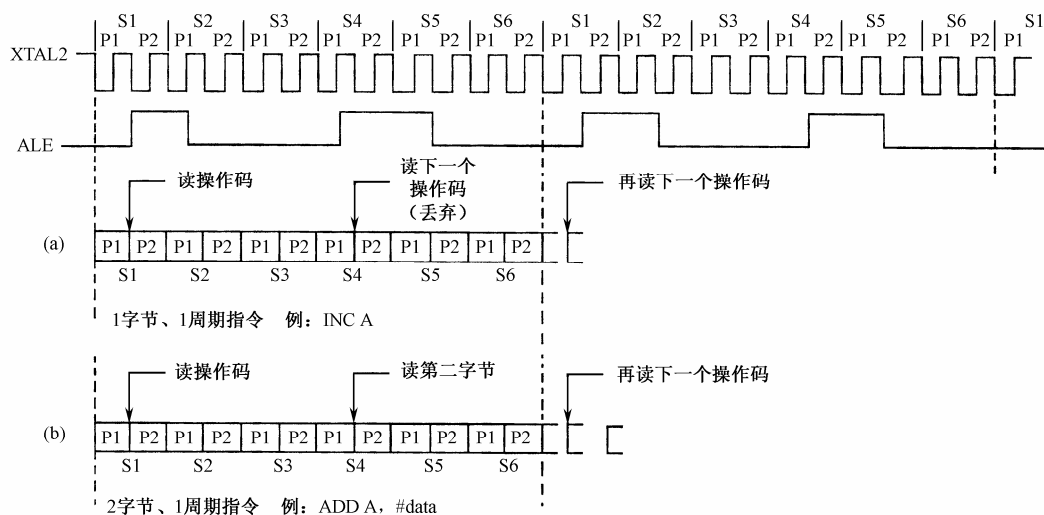


图 11-8 MCS-51 的取指/执行时序

单周期指令的执行始于 S1P2，此时操作码被锁存于指令寄存器内。若是双字节指令，则在同一机器周期的 S4 读第二字节。若是单字节指令，则在 S4 仍有读操作，但无效，且程序计数器 PC 不增量。图 11-8(a)和(b)分别给出了 1 字节、1 周期和 2 字节 1 周期指令的时序，都能在 S6P2 结束时完成操作。

## 11.5 定时/计数器

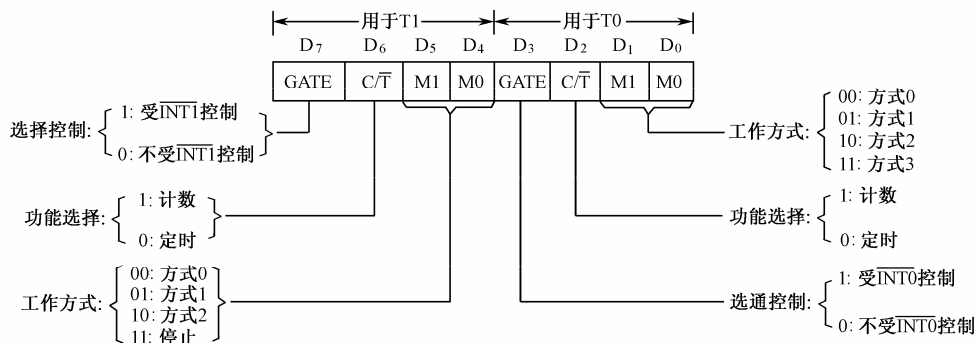
MCS-51 系列单片机内部提供了两个可编程的定时/计数器 T0 和 T1，它们可以用于定时或对外部脉冲（事件）计数，还可以作为串行口的波特率发生器。

### 1. 定时/计数器的控制字

定时/计数器 T0 和 T1 是可编程的，因此，在使用前必须对其初始化，CPU 向 TMOD 和 TCON 两个 8 位的特殊功能寄存器写入控制字，用来设置 T0 和 T1 工作方式和控制。

#### (1) TMOD

TMOD 为定时/计数器方式控制寄存器，其格式如下：



① 工作方式选择 M1、M0。T0、T1 的工作方式由 TMOD 中的 M1、M0 两位的状态确定，对应关系如表 11-6 所示。

表 11-6 定时/计数器工作方式的选择

M <sub>1</sub>	M <sub>0</sub>	工作方式	功 能 说 明
0	0	0	13 位定时/计数器
0	1	1	16 位定时/计数器
1	0	2	常数能够自动装入 8 位定时/计数器
1	1	3	把 T0 分为 2 个 8 位的定时/计数器

② 定时器和外部计数方式选择位  $C/\bar{T}$ 。 $C/\bar{T}=0$ ，定时/计数器为定时器方式。 $C/\bar{T}=1$  时，定时/计数器为计数方式。采用外部引脚 T0（或 T1）的输入脉冲作为计数脉冲。当 T0（或 T1）输入引脚脉冲信号发生由高到低的负跳变时，计数器进行加 1 计数。

③ 门控信号 GATE。GATE=1 时，定时/计数器的计数受外部引脚输入电平的控制： $\overline{INT0}$  控制 T0 运行， $\overline{INT1}$  控制 T1 运行。GATE=0 时，定时/计数器的运行不受外部输入引脚的控制。

## (2) TCON

控制寄存器 TCON 用于控制定时/计数器的启、停、溢出标志和外部中断信号触发方式。TCON 各位作用如下：

	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H
TCON (88H)	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1：T1 溢出标志位。当定时/计数器计满数产生溢出时，由硬件自动使 TF1 置 1，并向 CPU 申请中断。进入中断服务程序后，TF1 又被硬件自动清 0。TF1 也可作为程序查询的标志位，在查询方式下由软件清 0。

TR1：T1 运行控制位。TR1 由软件置 1，而由定时/计数器 T1 开始启动计数。软件使 TR1 清 0，定时/计数器 T1 停止工作。

TF0：T0 溢出标志位。其功能同 TF1。

TR0：T0 运行控制位。其功能同 TR1。

IE1、IT1、IE0、IT0：外部中断  $\overline{INT1}$  和  $\overline{INT0}$  请求方式控制位。

## 2. 定时/计数器工作方式

### (1) 工作方式 0

工作方式 0 为 13 位定时/计数器。此时，16 位寄存器 TH0、TL0（TH1、TL1）中，TH0（TH1）和 TL0（TL1）的低 5 位存放计数值，TL0（TL1）中的高三位不用，从而构成了 13 位计数。T0（或 T1）的方式 0 结构如图 11-9 所示。

由图 11-9 可以看出，当  $C/\bar{T}=0$ ，为定时方式时，多路开关与连接振荡器的 12 分频器输出连通，此时 T0 对机器周期进行计数。其定时时间为：

$$T = (2^{13} - X) \times 12 / f_{\text{osc}} = (2^{13} - X) \times \text{机器周期}$$

其中,  $X$  为计数初值。

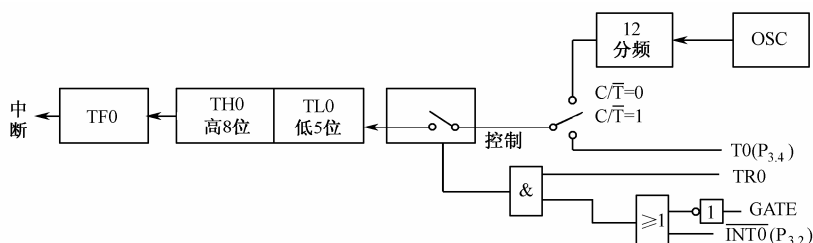


图 11-9 T0 (或 T1) 的方式 0 结构

### (2) 工作方式 1

工作方式 1 是 16 位定时/计数器, 其结构几乎与方式 0 完全相同, 惟一的区别是计数器的长度为 16 位。

### (3) 工作方式 2

定时/计数器工作方式 2 是一个能自动再装入初值的 8 位定时/计数器。图中 TL0 用做 8 位计数器, TH0 用做常数寄存器, 其内容可由软件预置。当 TL0 计数器满产生溢出时, 不仅置位 TF0 标志, 而且自动将 TH0 中的内容送至 TL0, 使定时/计数器从所置初值重新开始计数。在这种工作方式下, 用户不用在程序中重新装入初值。T0 (或 T1) 方式 2 的结构如图 11-10 所示。

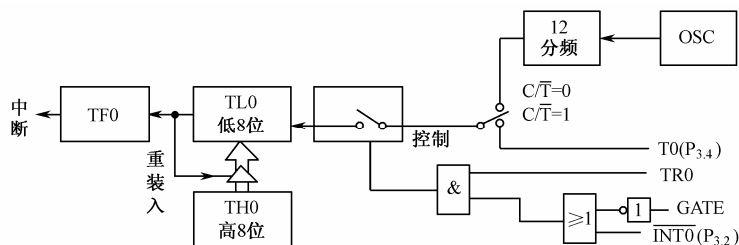


图 11-10 T0 (或 T1) 方式 2 的结构

### (4) 工作方式 3

工作方式 3 对 T0 和 T1 是大不相同的。

若将 T0 设置为方式 3, TL0 和 TH0 被分为两个互相独立的 8 位计数器。其中 TL0 用原 T0 的各控制位、引脚和中断源, 即  $C/\overline{T}$ , GATE、TR0、TF0 和 T0 ( $P_{3.4}$ )、 $\overline{INT0}$  ( $P_{3.2}$ ) 引脚。TL0 除仅用 8 位寄存器外, 其功能和操作与方式 0 (13 位计数器)、方式 1 (16 位计数器) 完全相同。TL0 也可设置为定时器方式或计数器方式。

TH0 只有简单的内部定时功能。它占用了定时器 T1 的控制位 TR1 和 T1 的中断标志位 TF1, 其启动和关闭仅受 TR1 的控制。T0 方式 3 的结构如图 11-11 所示。

定时器 T1 无工作方式 3 状态, 若将 T1 设置为方式 3, 就会使 T1 立即停止计数, 即保持原有的计数值, 其作用相当于使  $TR1=0$ , 封锁与门, 断开计数开关。

在一般情况下, 当定时器 T1 用做串行口波特率发生器时, 定时器 T0 才设置为工作方式 3。此时, 常把定时器 T1 设置为方式 2。

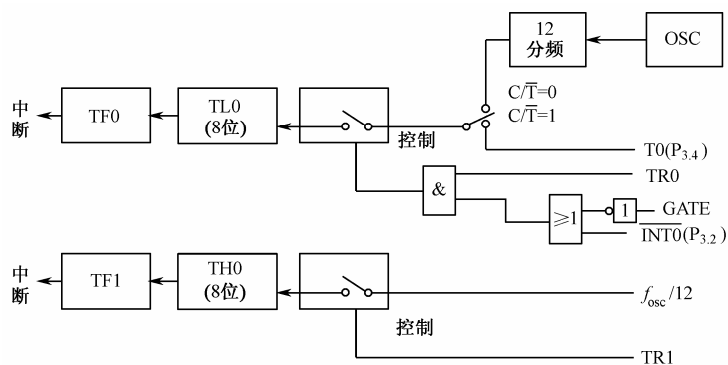


图 11-11 T0 方式 3 的结构

## 11.6 中断控制系统

### 11.6.1 中断系统结构

8051 单片机提供 5 个中断源。其中 2 个为外部中断请求  $\overline{\text{INT0}}$  和  $\overline{\text{INT1}}$ ；2 个为片内定时/计数器溢出中断请求 TF0 和 TF1；1 个为片内串行口中中断请求 TI 和 RI。MCS-51 单片机的中断系统由中断相关特殊功能寄存器、中断入口、查询逻辑电路等组成，其结构框图如图 11-12。

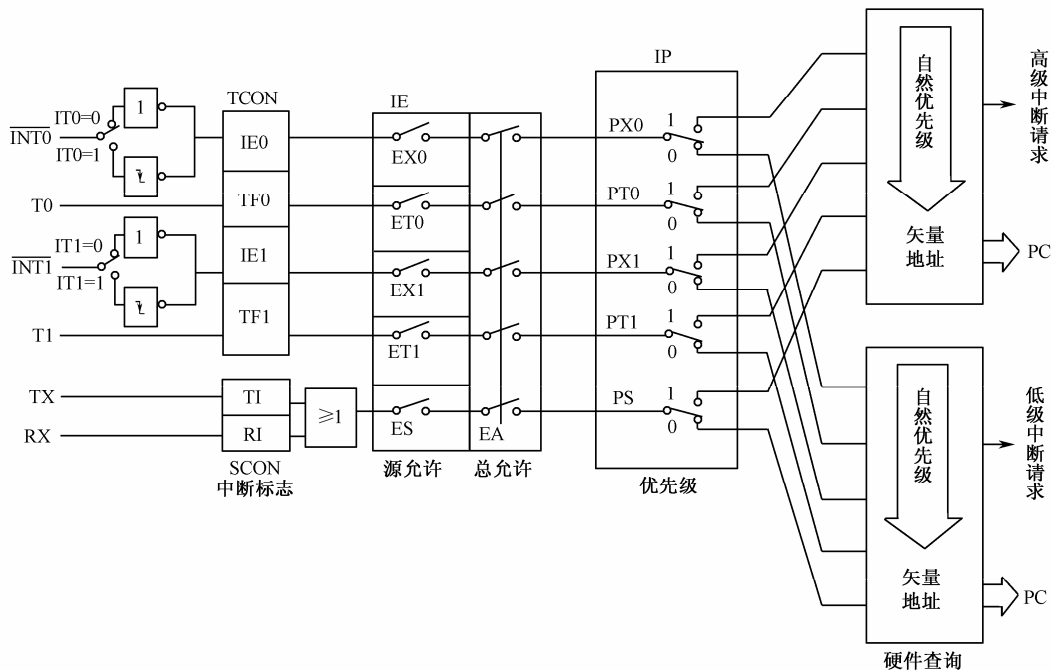


图 11-12 中断系统结构框图

### 1. 定时/计数器控制寄存器 TCON

TCON 为定时/计数器 T0、T1 的控制器，同时也锁存了 T0、T1 的溢出中断源和外部中断请求源。与中断有关的位如下：

	8FH		8DH		8BH		8AH	89H	88H
TCON(88)	TF1		TF0		IE1	IT1	IE0	IT0	

- IT0——外部中断 0 控制位。选择外部中断 0 为边沿触发方式或电平触发方式。

当 IT0=0 时，外部中断 0 为电平触发方式， $\overline{\text{INT0}}$  低电平有效。

当 IT0=1 时， $\overline{\text{INT0}}$  为边沿触发方式， $\overline{\text{INT0}}$  输入脚上由高到低的负跳变有效。IT0 可由软件置 1 或清 0。

- IE0——外部中断 0 的中断申请标志。

当 IT0=0，即为电平触发方式时，每个机器周期单片机采样  $\overline{\text{INT0}}$ ，若  $\overline{\text{INT0}}$  为低电平，则将 IE0 置 1，直接触发外部中断，否则 IE0 清 0。

当 IT0=1，即  $\overline{\text{INT0}}$  为边沿触发方式时，当 CPU 在第一个机器周期采样到  $\overline{\text{INT0}}$  为高电平，而第二个机器周期采样到  $\overline{\text{INT0}}$  为低电平时，则将 IE0 置 1。

IE0 为 1 表示外部中断 0 正在向 CPU 申请中断，当 CPU 响应中断，转向中断服务程序时，单片机内部硬件将 IE0 清 0。

- IT1——选择外部中断 1 为边沿触发方式或电平触发方式的控制位。其功能与 IT0 类似。

- IE1——外部中断 1 的中断申请标志。其功能与 IE0 类似。

- TF0——片内定时/计数器 0 溢出中断申请标志。

当启动 T0 开始计数后，定时/计数器 0 从初值开始进行加 1 计数。当最高位产生溢出时，由硬件将 TF0 置 1，向 CPU 申请中断；CPU 响应 TF0 中断时，将 TF0 清 0。此外，TF0 也可由软件清 0（查询方式）。

- TF1——片内定时/计数器 1 溢出中断申请标志。其功能类似 TF0。

当 8031 复位后，TCON 被清 0。

### 2. 串行口控制寄存器 SCON

SCON 的字节地址为 98H，它的低两位是串行口的接收中断和发送中断标志，其格式为：

	99H	98H
SCON(98H)	TI	RI

RI 和 TI：串行口内部表示中断申请标志。当串行口发送或接收完一帧数据时，将 SCON 中的 TI 或 RI 位置 1，向 CPU 申请中断。在 CPU 响应串行口的中断时，并不清 0 TI 和 RI 中断标志，TI 和 RI 必须由软件清 0。

## 11.6.2 中断系统的控制

### 1. 中断允许寄存器 IE

MCS-51 单片机中, 特殊功能寄存器 IE 为中断允许寄存器, 控制 CPU 对中断源的开放或屏蔽, 以及每个中断源是否允许中断。其格式为:

	AFH			ACH		ABH	AAH	A9H	A8H
IE(A8H)	EA	—	—	ES	ET1	EX1		ET0	EX0

- EA——CPU 中断开放标志位。EA=1, CPU 开放中断; EA=0, CPU 屏蔽所有的中断申请。
- ES——串行口中断允许位。ES=1, 允许串行口中断; ES=0, 禁止串行口中断。
- ET1——定时/计数器 T1 的溢出中断允许位。ET1=1, 允许 T1 中断; ET1=0, 禁止 T1 中断。
- EX1——外部中断 1 中断允许位。ET1=1, 允许外部中断 1 中断; ET1=0, 禁止外部中断 1 中断。
- ET0——定时/计数器 T0 的溢出中断允许位。ET0=1, 允许 T0 中断; ET0=0, 禁止 T0 中断。
- EX0——外部中断 0 中断允许位。EX0=1, 允许外部中断 0 中断; EX0=0, 禁止外部中断 0 中断。

MCS-51 单片机复位时, IE 将被清 0。

### 2. 中断源优先级设定寄存器 IP (B8H)

MCS-51 单片机有两个中断优先级, 对于每一个中断请求源可编程为高优先级中断或低优先级中断。它们可实现二级中断嵌套, 一个正在被执行的低优先级中断服务程序能被高优先级中断源所中断, 但不能被另一个低优先级中断源所中断。中断源优先级的控制位, 用户可用软件设定, 其格式如下:

				BCH	BBH	BAH	B9H	B8H
IP (B8H)	—	—	—	PS	PT1	PX1	PT0	PX0

- PS——串行口中断优先级控制位。PS=1, 设定串行口为高优先级中断; PS=0, 为低优先级中断。
- PT1——T1 中断优先级控制位。PT1=1, 设定定时器 T1 为高优先级中断; PT1=0, 为低优先级中断。
- PX1——外部中断 1 中断优先级控制位。PX1=1, 设定外部中断 1 为高优先级中断; PX1=0, 为低优先级中断。
- PT0——T0 中断优先级控制位。PT0=1, 设定定时器 T0 为高优先级中断; PT0=0, 为低优先级中断。
- PX0——外部中断 0 中断优先级控制位。PX0=1, 设定外部中断 0 为高优先级中断;



PX0=0，为低优先级中断。

当系统复位后，IP 各位均为 0，所有中断设置为低优先级中断。

当 CPU 同时收到几个同一优先级级别的中断请求时，哪一个中断请求将得到服务，取决于内部的硬件查询顺序，CPU 将按自然优先级顺序确定应该响应哪个中断请求。其自然优先级由硬件形成，排序如表 11-7 所示。

MCS-51 单片机具有五个中断源，每一个中断源有一个中断入口地址。中断源入口地址如表 11-8 所示。

表 11-7 中断源优先级排序

中 断 源	同级内部优先级
外部中断 0	最高级
定时器 T0 中断	
外部中断 1	
定时器 T1 中断	
串行口中断	最低级

表 11-8 中断源入口地址

中 断 源	中断矢量地址
外部中断 0 ( $\overline{\text{INT0}}$ )	0003H
定时器 T0 中断	000BH
外部中断 1 ( $\overline{\text{INT1}}$ )	0013H
定时器 T1 中断	001BH
串行口中断	0023H

## 11.7 串行口

### 1. 串行口的组成

MCS-51 串行口由串行数据缓冲器 SBUF、发送控制器、接收控制器、输入移位寄存器和输出控制门等部件组成。单片机的 P<sub>3.0</sub>（10 脚）和 P<sub>3.1</sub>（11 脚）分别为接收端 RXD 和发送端 TXD。串行口组成示意图如图 11-13 所示。

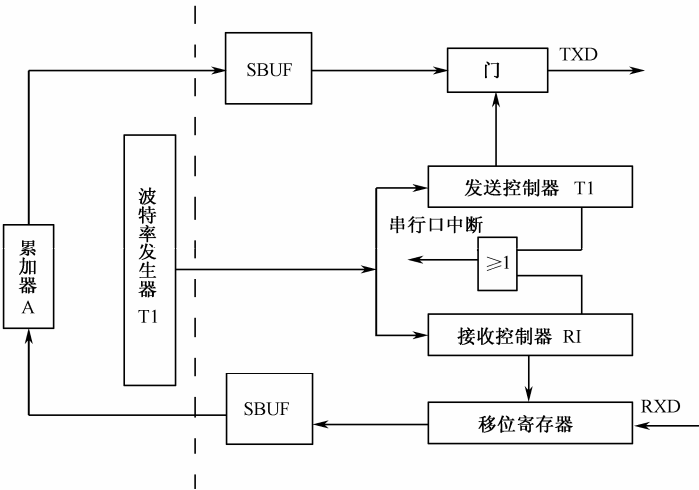


图 11-13 MCS-51 单片机串行口组成示意图

## 2. 串行口控制寄存器SCON

	D7	D6	D5	D4	D3	D2	D1	D0
SCON (98H)	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

- SM0、SM1——串行口的方式选择位。其编码如表 11-9 所示。

表 11-9 串行口的工作方式

方 式	SM0	SM1	功 能 说 明
0	0	0	移位寄存器方式，波特率为 $f_{osc}/12$
1	0	1	8 位 UART，波特率可变 ( $T=溢出率/n$ )
2	1	0	9 位 UART，波特率为 $f_{osc}/64$ 或 $f_{osc}/32$
3	1	1	9 位 UART，波特率可变 ( $T=溢出率/n$ )

表中  $f_{osc}$  为晶振频率；UART 为通用异步接收和发送器。

- REN——允许接收控制位。用软件置 1 或清 0，REN=1 时，允许接受；若 REN=0，则禁止接收。

- SM2——允许方式 2 和方式 3 进行多机通信的控制位。

在串行口工作方式 0 时，SM2 必须是 0，不用 TB8 和 RB8。

在串行口工作方式 1 的情况下，当 SM2=0 时，RB8 是接收到的停止位，若 SM2=1，则只有收到有效的停止位才会激活接收中断 RI；若没有接收到有效停止位，则 RI 清 0。

在方式 2 或方式 3 中，TB8 是要发送的第 9 位数据，可用软件置 1 或置 0。RB8 是接收到的第 9 位数据。若 SM2=1，而接收到的第 9 位数据 (RB8) 为 0，则 RI (接收中断) 不被激活，利用这个特点可以进行多机通信。

- TB8——在方式 2 或方式 3 中要发送的第 9 位数据，根据需要可用软件置位或复位。

- RB8——在方式 2 或方式 3 中，是所接收到的第 9 位数据。在方式 1 中，若 SM2=0，则 RB8 是接收到的停止位。在方式 0 中，不使用 RB8。

- TI——发送中断标志位。在工作方式 0 中，发送完 8 位数据后，由硬件置 1，向 CPU 申请发送中断。CPU 响应中断后，必须用软件清 0。在其他工作方式中，它在停止位开始发送时由硬件置 1，同样必须用软件清 0。

- RI——接收中断标志位。在工作方式 0 中，接收完 8 位数据后，由硬件置 1，向 CPU 申请接收中断，CPU 响应中断后，必须用软件清 0。在其他工作方式中，在接收到停止位的中间时刻由硬件置 1，向 CPU 申请中断，表示一帧数据接收结束，并已装入缓冲器，要求 CPU 取走数据。CPU 响应中断，取走数据后必须由软件清 0，解除中断请求，准备接收下一帧数据。

### 3. 专用寄存器PCON和波特率的选择

PCON 是为了在 MCS-51 单片机上实现电源控制而设置的，其中只有一位 SMOD 与串行口工作有关。它的格式和功能如下：

PCON	D7	D6	D5	D4	D3	D2	D1	D0
(87)	SMOD	—	—	—	—	—	—	—

SMOD 称为波特率选择位。在工作方式 1、工作方式 2 和工作方式 3 时，若 SMOD=1，则波特率提高一倍；若 SMOD=0，则波特率不加倍。

由上面讨论可知，串行口方式 0 的波特率是固定不变的，为  $f_{\text{osc}}/12$ ；方式 2 的波特率也只有两种，为  $f_{\text{osc}}/64$  或  $f_{\text{osc}}/32$ 。

下面讨论串行口方式 1 与串行口方式 3 的波特率设定。

$$\text{波特率} = (\text{定时器 T1 的溢出率})/n, \quad n = \begin{cases} 32 & (\text{SMOD} = 0) \\ 16 & (\text{SMOD} = 1) \end{cases}$$

定时/计数器的溢出率取决于自动装载方式 2，TL1 作计数用，自动重装的值放在 TH1 中。溢出率可由下式确定：

$$\text{溢出率} = \text{计数速率} / [256 - (\text{TH1})]$$

若波特率指定就可以计算出定时器的溢出率，根据溢出率就可以确定定时器 T1 的预置值。

【例 11-1】设振荡频率  $f_{\text{osc}}=12\text{MHz}$ ，SMOD=0，选定波特率为 1200，确定定时器 T1 的预置值。

SMOD=0， $n=32$  时，定时器 T1 的溢出率为：

$$\text{定时器 T1 的溢出率} = \text{波特率} \times n = 1200 \times 32 = 38400$$

当定时/计数器设定为定时方式时，计数速率为  $f_{\text{osc}}/12$ 。根据溢出率的计算公式可以计算出 TL1 级 TH1 的预置值。其计算公式如下：

$$\text{TH1} = 256 - \text{计数速率} / \text{溢出率} = 256 - 12 \times 10^6 / 1200 \times 32 \times 12 = 256 - 24 = 232 = \text{E8H}$$

## 11.8 MCS-51 单片机指令系统

MCS-51 单片机共有 111 条指令，按功能可分为五类：① 数据传送类指令 29 条；② 算术运算类指令 24 条；③ 逻辑操作类指令 24 条；④ 位操作类指令 17 条；⑤ 控制转移类指令 17 条。

### 11.8.1 寻址方式

寻址方式就是确定操作数单元地址或下一条要执行指令的地址的方法。

#### 1. 立即寻址

立即寻址是指指令中直接给出操作数的寻址方式。例如：

MOV A, #20H ; 将立即数#20H 送入累加器 A 中  
MOV DPTR, #4000H ; 将立即数#4000H 送入 16 位 DPTR 数据指针中

“#” 标志表示为立即数。

#### 2. 直接寻址

直接寻址是指指令中给出操作数地址的寻址方式。例如：

MOV A, 32H ; 将内部 RAM 的 32 单元中数据送给累加器 A

### 3. 寄存器寻址

寄存器寻址就是操作数在寄存器中，因此指定了寄存器就可得到操作数。例如：

MOV A, R7 ; 将寄存器 R7 的内容送给累加器 A

### 4. 寄存器间接寻址

寄存器间接寻址是将指令指定的寄存器内容作为地址，对该地址单元中的内容进行操作的寻址方式。例如：

MOV A, @R0 ; 将 R0 中的内容作为地址单元中的内容，送给累加器 A

寄存器前的标志“@”表示间接寄存器。MCS-51 指令系统中规定 R0、R1 和 DPTR 可作为间接寻址的寄存器。

### 5. 相对寻址

相对寻址是以当前 PC 值加上指令中给出的相对偏移量形成程序转移的目的地址的寻址方式。例如：

JC 08H ; 该指令表示 CY=0 则不跳转

若 CY=1，则以当前的 PC 值为基址加上偏移量 08H 后，所得到的结果为该转移指令的目的地址。

### 6. 变址寻址

是以 DPTR 或 PC 作为基址寄存器，以累加器 A 作为变址寄存器，并以两者内容相加形成的 16 位地址作为操作数地址。例如：

MOVC A, @A+DPTR ; 把 DPTR 和 A 的内容相加作为 16 位程序存储器地址，再把该地址单元的内容送给累加器 A

### 7. 位寻址

位寻址是指片内 RAM 的位寻址区和某些可位寻址的特殊功能寄存器进行位操作时的寻址方式。例如：

ANL C, 30H

该指令的功能是 C 的状态和地址为 30H 的位状态进行逻辑与操作，并把结果保存在 C 中。

## 11.8.2 指令格式及说明

### 1. 指令格式

MCS-51 单片机汇编语言指令通常由操作码助记符和操作数两部分组成。指令格式如下：

[标号]: 操作码 [目的操作数], [源操作数]; 注释

标号表示该指令的符号地址；操作码表示该指令的操作功能；操作数表示参加操作的数据来源和操作结果存放在什么地方；注释部分是为该条指令所做的说明以便于阅读，该部分也可以省略。

### 2. 指令中符号的注释说明

在分类介绍指令之前，先把指令中使用的一些符号的意义做简单说明。

Rn——当前寄存器组的8个通用寄存器R0~R7，所以 $n=0\sim7$ 。

Ri——可用做间接寻址的寄存器，只能是R0、R1两个寄存器，所以 $I$ 为0或1。

data——内部RAM的8位地址。既可以是内部RAM的低128个单元地址，也可以是专用寄存器的单元地址或符号。因此在指令中data表示直接寻址方式。

#data——8位立即数。

#data16——16位立即数。

Addr16——16位目的地址，只限于在LCALL和LJMP指令中使用。

Addr11——11位目的地址，只限于在ACALL和AJMP指令中使用。

rel——相对转移指令中的偏移量，为8位带符号补码数。

DPTR——数据指针。

bit——内部RAM（包括专用寄存器）中的直接寻址位。

A——累加器。

B——寄存器。

C——进位标志位，它是布尔处理机的累加器，也称之为累加位。

@——间址寄存器的前缀标志。

/——加在位地址的前面，表示对该位状态取反。

( $\times$ )——某寄存器或某单元的内容。

(( $\times$ ))——由 $\times$ 间接寻址的单元中的内容。

←——箭头左边的内容被箭头右边的内容所取代。

### 11.8.3 数据传送类指令

#### 1. 以累加器A为目的操作数的指令

```
MOV  A, Rn          ; (A)←(Rn)
MOV  A, #data       ; (A)←#data
MOV  A, data        ; (A)←(data)
MOV  A, @Ri         ; (A)←((Ri))
```

#### 2. 以Rn为目的操作数的指令

```
MOV  Rn, data       ; (Rn)←(data)
MOV  Rn, A          ; (Rn)←(A)
MOV  Rn, #data      ; (Rn)←#data
```

#### 3. 以直接地址为目的操作数的指令

```
MOV  data, A        ; (data)←(A)
MOV  data, Rn       ; (data)←(Rn)
MOV  data, data     ; (data)←(data)
MOV  data, @Ri      ; (data)←((Ri))
MOV  data, #data    ; (data)←#data
```

#### 4. 以间接地址为目的操作数的指令

```
MOV  @Ri, A         ; ((Ri))←(A)
MOV  @Ri, data      ; ((Ri))←(data)
```

MOV @Ri, #data ; ((Ri))←#data

### 5. 堆栈操作指令

MCS-51 单片机片内的 RAM 中, 设定一个后进先出区域作为堆栈, 专用寄存器中有一个堆栈指针 SP, 它指出栈顶的位置。指令系统中有两条用于数据传送的栈操作指令。

PUSH data ; (sp)←(sp)+1, ((sp))←(data)

POP data ; (data)←((sp)), (sp)←(sp)-1

### 6. 累加器A与外部数据存储器传送指令

MOVX A, @DPTR ; (A)←((DPTR)) 读外部 RAM 或 I/O 口

MOVX A, @Ri ; (A)←((Ri)) 读外部 RAM 或 I/O 口

MOVX @DPTR, A ; ((DPTR))←(A) 写外部 RAM 或 I/O 口

MOVX @Ri, A ; ((Ri))←(A) 写外部 RAM 或 I/O 口

### 7. 查表指令

MOVC A, @A+PC ; (PC)←(PC)+1, (A)←((A)+(PC))

MOVC A, @A+DPTR ; (A)←((A)+(DPTR))

### 8. 数据交换指令

XCH A, Rn ; (A)↔(Rn)

XCH A, @Ri ; (A)↔((Ri))

XCH A, data ; (A)↔(data)

XCHD A, @Ri ; (A<sub>3~0</sub>)↔((Ri<sub>3~0</sub>))

SWAP A ; (A<sub>3~0</sub>)↔(A<sub>7~4</sub>)

### 9. 传送指令应用举例

【例 11-2】把立即数 38H 送到内部 RAM 的 25H 地址单元中。

MOV 25H, #38H ; (25H)←#38H

【例 11-3】把立即数 38H 送到内部 RAM 的 25H 地址单元中。

MOV R0, #25H ; (R0)←#25H

MOV @R0, #38H ; ((R0))←#38H

【例 11-4】把立即数 38H 送到内部 RAM 的 25H 地址单元中。

MOV A, #38H ; (A)←#38H

MOV R0, #25H ; (R0)←#25H

MOV @R0, A ; ((R0))←(A)

【例 11-5】把立即数#27H 传送到外部 RAM 的 2000H 地址单元中。

MOV A, #27H ; (A)←#27H

MOV DPTR, #2000H ; (DPTR)←#2000H

MOVX @DPTR, A ; ((DPTR))←(A)

【例 11-6】已知(SP)=62H, (62H)=70H, (61H)=30H, 执行下列程序:

POP DPH ; (DPH)←((SP)), (SP)←(SP)-1

POP DPL ; (DPL)←((SP)), (SP)←(SP)-1

结果为(DPTR)=7030H, (SP)=60H。

【例 11-7】已知(R0)=20H, (A)=3FH, (20H)=75H, 执行下列指令:

XCHD A, @R0

结果为(R0)=20H, (A)=35H, (20H)=7F。

### 11.8.4 算术运算类指令

此类指令包括加、减、乘、除、加 1、减 1 等指令。这类指令大都影响标志位。加法和减法的执行结果将影响 PSW 的 CY 进位位、OV 溢出位、AC 辅助进位位、P 奇偶校验位；乘除法指令的执行结果影响 OV 溢出位，CY 进位位和 P 奇偶校验位。加 1、减 1 指令执行结果影响 P 奇偶校验位。

#### 1. 加法指令

##### (1) 不带进位的加法指令

ADD	A, Rn	; (A)←(A)+(Rn)
ADD	A, data	; (A)←(A)+(data)
ADD	A, @Ri	; (A)←(A)+((Ri))
ADD	A, #data	; (A)←(A)+#data

##### (2) 带进位加法指令

ADDC	A, Rn	; (A)←(A)+(Rn)+(C)
ADDC	A, data	; (A)←(A)+(data)+(C)
ADDC	A, @Ri	; (A)←(A)+((Ri))+ (C)
ADDC	A, #data	; (A)←(A)+#data+(C)

#### 2. 带借位减法指令

SUBB	A, Rn	; (A)←(A)-(Rn)- (C)
SUBB	A, data	; (A)←(A)-(data)- (C)
SUBB	A, @Ri	; (A)←(A)-((Ri))- (C)
SUBB	A, #data	; (A)←(A)-#data - (C)

#### 3. 加 1 指令

INC	A	; (A)←(A)+1
INC	Rn	; (Rn)←(Rn)+1
INC	data	; (data)←(data)+1
INC	@Ri	; ((Ri)←((Ri))+1
INC	DPTR	; (DPTR)←(DPTR)+1

#### 4. 减 1 指令

DEC	A	; (A)←(A)-1
DEC	Rn	; (Rn)←(Rn)-1
DEC	data	; (data)←(data)-1
DEC	@Ri	; ((Ri)←((Ri))-1

#### 5. 十进制调整指令

十进制调整指令 DAA 用来对 BCD 码的加法运算结果进行修正。

执行该指令，修正的条件和方法：

若 $(A_{0\sim3}) > 9$  或 $(AC)=1$ ，则 $(A_{0\sim3})+6 \rightarrow (A_{0\sim3})$ ；若 $(A_{4\sim7}) > 9$  或 $(CY)=1$ ，则 $(A_{4\sim7})+6 \rightarrow (A_{4\sim7})$ 。

## 6. 乘法指令

MUL AB ;  $(A) \times (B) \rightarrow (B)_{15\sim8}, (A)_{7\sim0}$

## 7. 除法指令

DIV AB ;  $(A)/(B) \rightarrow (A)$ 商、 $(B)$ 余数

## 8. 编程举例

【例 11-8】假定有两个 4 字节的二进制数分别放在 20H 和 30H 为起始地址的单元中（先存低位），求这两个数的和，并将和数存放在 20H 为起始地址的单元中去。

```

ORG 2000H
MOV R0, #20H      ; 指向加数最低位
MOV R1, #30H      ; 指向另一个加数最低位
MOV R2, #04H      ; 字节数作计数值
CLR C              ; 清进位标志
LOOP: MOV A, @R0   ; 取加数的第一个字节
ADDC A, @R1        ; 加上另一个加数的一个字节
MOV @R0, A         ; 存和数
INC R0             ; 修改指针
INC R1
DJNZ R2, LOOP      ; 没加完循环
RET                ; 子程序返回

```

【例 11-9】多字节减法。已知被减数存放在以 30H 为起始地址的单元中（先存低位），减数存放在以 40H 为起始地址的单元中，差存放在 30H 为起始地址的单元中，每个数的字节长度为 4。

```

ORG 2000H
MOV R0, #30H      ; 指向被减数最低位
MOV R1, #40H      ; 指向减数最低位
MOV R2, #04H      ; 每个数的长度作计数初值
CLR C              ; 清进位标志
LOOP: MOV A, @R0   ; 取被减数
SBBB A, @R1        ; 完成一个字节减法运算
MOV @R0, A         ; 存差
INC R0             ; 修改地址指针
INC R1
DJNZ R2, LOOP      ; 没减完循环
RET                ; 子程序返回

```



## 11.8.5 逻辑运算指令

逻辑操作类指令包括与、或、异或、求反、移位指令等。

### 1. 逻辑与指令

ANL	A, Rn	; $(A) \leftarrow (A) \wedge (Rn)$
ANL	A, data	; $(A) \leftarrow (A) \wedge (data)$
ANL	A, @Ri	; $(A) \leftarrow (A) \wedge ((Ri))$
ANL	A, #data	; $(A) \leftarrow (A) \wedge \#data$
ANL	data, A	; $(data) \leftarrow (data) \wedge (A)$
ANL	data, #data	; $(data) \leftarrow (data) \wedge \#data$

### 2. 逻辑或指令

ORL	A, Rn	; $(A) \leftarrow (A) \vee (Rn)$
ORL	A, data	; $(A) \leftarrow (A) \vee (data)$
ORL	A, @Ri	; $(A) \leftarrow (A) \vee ((Ri))$
ORL	A, #data	; $(A) \leftarrow (A) \vee \#data$
ORL	data, A	; $(data) \leftarrow (data) \vee (A)$
ORL	data, #data	; $(data) \leftarrow (data) \vee \#data$

### 3. 逻辑异或指令

XRL	A, Rn	; $(A) \leftarrow (A) \oplus (Rn)$
XRL	A, data	; $(A) \leftarrow (A) \oplus (data)$
XRL	A, @Ri	; $(A) \leftarrow (A) \oplus ((Ri))$
XRL	A, #data	; $(A) \leftarrow (A) \oplus \#data$
XRL	data, A	; $(data) \leftarrow (data) \oplus (A)$
XRL	data, #data	; $(data) \leftarrow (data) \oplus \#data$

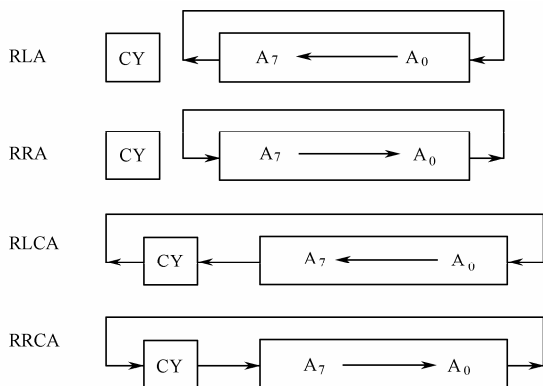
### 4. 求反指令

CPL	A	; $(A) \leftarrow (\bar{A})$
-----	---	------------------------------

### 5. 清0指令

CLR	A	; $(A) \leftarrow 0$
-----	---	----------------------

### 6. 循环移位指令



## 7. 编程举例

【例 11-10】对累加器 A 中的内容乘 8 操作。设(A)=01H。

编程如下：

```
RL    A    ; (A)←02H
RL    A    ; (A)←04H
RL    A    ; (A)←08H
```

【例 11-11】对累加器 A 中的内容除 8 操作。设(A)=08H。

编程如下：

```
RR    A    ; (A)←04H
RR    A    ; (A)←02H
RR    A    ; (A)←01H
```

【例 11-12】设 P1 中的内容为 AAH，累加器 A 中的内容为 10H。

执行下列程序：

```
ANL   P1, #0F0H ; (P1)=A0H
ORL   P1, #0FH  ; (P1)=AFH
XRL   P1, A      ; (P1)=BFH
```

## 11.8.6 位操作指令

### 1. 位传送指令

```
MOV    C, bit      ; (C)←(bit)
MOV    bit, C       ; (bit)←(C)
```

### 2. 位置位、复位指令

```
CLR    C            ; (C)←0
CLR    bit          ; (bit)←0
SETB   C            ; (C)←1
SETB   bit          ; (bit)←1
```

### 3. 位运算指令

```
ANL    C, bit       ; (C)←(C)∧(bit)
ANL    C, /bit      ; (C)←(C)∧( $\overline{\text{bit}}$ )
ORL    C, bit       ; (C)←(C)∨(bit)
ORL    C, /bit      ; (C)←(C)∨( $\overline{\text{bit}}$ )
CPL    C            ; (C)←( $\overline{C}$ )
CPL    bit          ; (bit)←( $\overline{\text{bit}}$ )
```

### 4. 位条件转移指令

以 C 为状态条件的转移指令：

```
JC    rel
```

(C)=1 转移指令，其转移控制为：

若(C)=1，则 $(pc) \leftarrow (pc) + 2 + rel$ ；若(C)=0，则 $(pc) \leftarrow (pc) + 2$ ，顺序执行。

## 5. 编程举例

【例 11-13】将 20H 位地址单元中的内容传送到 50H 位地址单元中。

```
MOV    C, 20H      ; (C) ← (20H)
MOV    50, C        ; (50H) ← (C)
```

【例 11-14】E、B、D 代表位地址，进行 E、B 内容的异或操作，即  $D = E \oplus B = \overline{E}B + E\overline{B}$ 。

编程如下：

```
MOV    C, B
ANL    C, /E        ; (C) ←  $\overline{E}B$ 
MOV    D, C
MOV    C, E
ANL    C, /B        ; (C) ←  $\overline{E}B$ 
ORL    C, D          ;  $\overline{E}B + E\overline{B}$ 
MOV    D, C          ; 异或结果送 D 位
```

## 11.8.7 控制转移类指令

### 1. 无条件转移指令

```
LJMP    addr16      ; (pc) ← addr16
AJMP    addr11      ; (pc) ← (pc) + 2, (pc10~0) ← addr11, (pc15~11) 不变
SJMP    rel         ; (pc) ← (pc) + 2 + rel
JMP      @A+DPTR     ; (pc) ← (A) + (DPTR)
```

### 2. 条件转移指令

#### (1) 判断累加器 A 结果转移指令

```
JZ      rel         ; 若(A)=0, 则(pc) ← (pc) + 2 + rel, 若(A) ≠ 0, 则(pc) ← (pc) + 2
JNZ     rel         ; 若(A) ≠ 0, 则(pc) ← (pc) + 2 + rel, 若(A) = 0, 则(pc) ← (pc) + 2
```

#### (2) 比较转移指令

```
CJNE    A, data, rel ; 若(A)=(data), (pc) ← (pc) + 3, (C) ← 0
                        若(A) ≠ (data), (pc) ← (pc) + 3 + rel
                        若(A) > (data), (C) ← 0; (A) < (data), (C) ← 1
CJNE    A, #data, rel ; 若(A)=#data, (pc) ← (pc) + 3, (C) ← 0
                        若(A) ≠ #data, (pc) ← (pc) + 3 + rel
                        若(A) > #data, (C) ← 0; (A) < #data, (C) ← 1
CJNE    Rn, #data, rel ; 若(Rn)=#data, (pc) ← (pc) + 3, (C) ← 0
                        若(Rn) ≠ #data, (pc) ← (pc) + 3 + rel
                        若(Rn) > #data, (C) ← 0; (Rn) < #data, (C) ← 1
```

CJNE @Ri, #data, rel ; 若((Ri))=#data, (pc)←(pc)+3, (C)←0  
                           若((Ri))≠#data, (pc)←(pc)+3+rel  
                           若((Ri))>#data, (C)←0; ((Ri))<#data, (C)←1

### (3) 循环转移指令

DJNZ Rn, rel ; (Rn)←(Rn)-1  
                   若(Rn)=0, 则(pc)←(pc)+2  
                   若(Rn)≠0, 则(pc)←(pc)+2+rel  
  
 DJNZ data, rel ; (data)←(data)-1  
                   若(data)=0, 则(pc)←(pc)+3  
                   若(data)≠0, 则(pc)←(pc)+3+rel

## 3. 调用子程序及返回指令

### (1) 绝对调用指令

ACALL addr11 ; (pc)←(pc)+2, (sp)←(sp)+1  
                   ((sp))←(pc<sub>7-0</sub>), (sp)←(sp)+1  
                   ((sp))←(pc<sub>15-8</sub>), (p<sub>10-0</sub>)←addr11

### (2) 长调用指令

LCALL addr16 ; (pc)←(pc)+3  
                   (sp)←(sp)+1  
                   ((sp))←(pc<sub>7-0</sub>)  
                   (sp)←(sp)+1  
                   ((sp))←(pc<sub>15-8</sub>)  
                   (pc)←addr16

### (3) 子程序返回指令

RET ; (pc<sub>15-8</sub>)←((sp))  
           (sp)←(sp)-1  
           (pc<sub>7-0</sub>)←((sp))  
           (sp)←(sp)-1

### (4) 中断返回指令

RETI ; (pc<sub>15-8</sub>)←((sp))  
           (sp) ←(sp)-1  
           (pc<sub>7-0</sub>)←((sp))  
           (sp) ←(sp)-1

## 4. 编程举例

【例 11-15】温度控制系统，采集的温度值 (Ta) 放在累加器 A 中。此外，内部 RAM 54H 单元存放温度下限值 (T54)，在 55H 单元存放温度上限值 (T55)。若 Ta>T55，程序转向 JW (降温处理程序)；若 Ta<T54，则程序转向 SW (升温处理程序)；若 T55≥Ta≥T54，则程序转向 FH (返回主程序)。编程如下：

```

                CJNE    A, 55H, Loop1    ; Ta≠T55 转向 Loop1
                AJMP    FH              ; Ta=T55 返回
Loop1:          JNC     JW              ; 若(C)=0, 表明 Ta>T55, 转降温处理程序
                CJNE    A, 54H, Loop2    ; Ta≠T54 转向 Loop2
                AJMP    FH              ; Ta=T54 返回
Loop2:          JCSW                    ; 若(C)=1 表示 Ta<T54, 转升温处理程序
FH :           RET
    
```

## 11.9 应用举例

【例 11-16】应用定时器 T0 产生 1ms 定时，并使 P<sub>1.0</sub> 输出周期为 2ms 的方波，已知晶体 6MHz。

设定定时器的计数初值为 X， $(2^{13}-X) \times 2 \times 10^{-6} = 1 \times 10^{-3}$ ， $X = 7692$ 。

13 位二进制数表示为  $X=1111000001100$ ，TH0=0F0H，TL0=0CH。

利用查询 TF0 状态来控制 P<sub>1.0</sub> 端输出周期为 2ms 的方波。

程序设计如下。

```

                ORG     2000H
                MOV     TMOD, #00H      ; 写入方式控制字
                MOV     TL0, #0CH       ; 计数初值写入
                MOV     TH0, #0F0H
                SETB    TR0              ; 启动 T0
LOOP:           JBC     TF0, PE          ; TF0=1 溢出转移 PE, 同时清除 TF0
                AJMP    LOOP            ; 没有溢出
PE:             MOV     TL0, #0CH       ; 重装计数初值
                MOV     TH0, #0F0H
                CPL     P1.0            ; 求反
                AJMP    LOOP            ; 无条件转移 LOOP
                END
    
```

【例 11-17】用定时器 T1 产生一个 25Hz 方波，由 P<sub>1.0</sub> 输出，采用查询方式进行控制，设定晶体频率 12MHz。

分析：25Hz 方波，周期为  $1/25 = 40\text{ms}$ ，采用定时器 T1 定时 20ms，将 P<sub>1.0</sub> 取反一次，即可得到 25Hz 的方波信号。

设定 20ms 的计数初值为 X，则有  $T = (2^{16}-X) \times 1 \times 10^{-6} = 20 \times 10^{-3}$ ， $X = 45536 = \text{B1E0H}$ 。

程序设计如下：

```

                ORG     2000H
                MOV     TMOD, #10H      ; T1 定时功能工作方式 1
                MOV     TH1, #0B1H      ; 写入初值
    
```

```

MOV     TL1, #0E0H
SETB    TR1                      ; 启动 T1
LOOP:   JBC    TF1, LP            ; TF1=1, 溢出转移, 同时 TF1 清零
        AJMP   LOOP
LP:      MOV    TH1, #0B1H        ; 重装初值
        MOV    TL1, #0E0H
        CPL    P1.0              ; P1.0 取反
        SJMP   LOOP
END

```

【例 11-18】由 P<sub>3.4</sub> 引脚 (T<sub>0</sub>) 输入一个低频脉冲信号 (其频率小于 0.5kHz), 要求 P<sub>3.4</sub> 每发生一次负跳变时, P<sub>1.0</sub> 输出一个 200 $\mu$ s 的同步负脉冲, 同时 P<sub>1.1</sub> 输出一个 400 $\mu$ s 的同步正脉冲。已知  $f_{\text{osc}}=6\text{MHz}$ 。

解: 按题意画出信号的波形如图 11-14 所示。

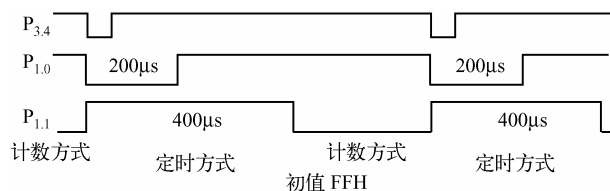


图 11-14 波形示意图

编程思路: 设初态 P<sub>1.0</sub> 输出高电平 (系统复位时即为高), P<sub>1.1</sub> 输出低电平, 设 T<sub>0</sub> 为方式 2, 计数工作方式 (初值为 FFH)。当加在 P<sub>3.4</sub> 上的外部脉冲产生由 1→0 的负跳变时, 则使 T<sub>0</sub> 计数器加 1 而产生溢出, 程序查询到 TF<sub>0</sub> 为 1 时, 改变为 200 $\mu$ s 定时工作方式, 并且使 P<sub>1.0</sub> 输出为 0, P<sub>1.1</sub> 输出为 1。当 T<sub>0</sub> 第一次定时 200 $\mu$ s 到时, 计数器溢出后, 使 P<sub>1.0</sub> 恢复为 1。T<sub>0</sub> 继续第二次 200 $\mu$ s 定时的计数产生溢出后恢复 P<sub>1.1</sub> 为 0。然后 T<sub>0</sub> 又恢复对外部脉冲的计数方式, 如此循环。

200 $\mu$ s 定时的计数初值为:  $X = 256 - 200 \times 6/12 = 156$ 。

程序如下:

```

START:  MOV    TMOD, #06H        ; T0 方式 2, 计数方式
        MOV    TH0, #0FFH       ; 计数初值
        MOV    TL0, #0FFH
        CLR    P1.1             ; P1.1 初态为 0
        SETB   TR0              ; 启动 T0
LOOP:   JBC    TF0, LP1         ; 检测外部信号负跳变
        SJMP   LOOP             ; 等待
LP1:    CLR    TR0              ; 关定时器
        MOV    TMOD, #02H       ; T0 改变为定时 200μs 方式 2
        MOV    TH0, #156        ; 定时的计数初值
        MOV    TL0, #156

```

```

                SETB    P1.1                ; P1.1 输出为 1
                CLR     P1.0                ; P1.0 输出 0
                SETB    TR0                 ; 启动 T0 定时
LOOP1:  JBC      TF0, LP2                   ; 第一个 200μs 到否?
                SJMP     LOOP1              ; 未到等待
LP2:    SETB     P1.0                      ; 到了 P1.0 恢复
LOOP2:  JBC      TF0, LP3                   ; 第二个 200 μs 到否?
                SJMP     LOOP2
LP3:    CLR      P1.1                      ; P1.0 恢复 0
                CLR     TR0                 ; 关定时器
                AJMP     START
    
```

【例 11-19】 应用 T0 方式 3，分别设定 200μs 和 400μs 定时并使 P<sub>1.0</sub> 和 P<sub>1.1</sub> 分别产生周期为 400 μs 和 800 μs 方波，已知晶体为 6MHz。本题采用中断控制方式。

解：定时 200μs 计数初值：

$$(2^8 - X) \times 2 \times 10^{-6} = 200 \times 10^{-6} \quad X = 156 = 9CH$$

定时 400 μs 计数初值：

$$(2^8 - X) \times 2 \times 10^{-6} = 400 \times 10^{-6} \quad X = 56 = 38H$$

程序设计：

```

                ORG      2000H
START:  AJMP     MAIN
                ORG      000BH
                AJMP     PIT0                ; 转 T0 中断处理入口
                ORG      001BH
                AJMP     PIT1                ; 转 T1 中断处理入口
                ORG      2100H
MAIN:   MOV      SP, #60H
                MOV      TMOD, #03H         ; 置方式 3
                MOV      TL0, #9CH          ; 定时 200 μs 计数初值
                MOV      TH0, #38H          ; 定时 400 μs 计数初值
                MOV      TCON, #50H         ; 启动 TL0、TH0 计数
                MOV      IE, #8AH          ; 中断允许 T0、T1 开放中断
LOOP:   AJMP     LOOP                        ; 等待中断
PITO:   MOV      TL0, #9CH                  ; T0 中断处理程序
                CPL      P1.0
                RETI
PIT1:   MOV      TH0, #38H                  ; T1 中断处理程序
                CPL      P1.1
                RETI
    
```

【例 11-20】利用 T0 门控位 GATE 来测试由  $\overline{\text{INT0}}$  引脚输入的正脉冲宽度，已知  $f_{\text{osc}}=12\text{MHz}$ ，所测得的高 8 位值存入片内 RAM 的 21H 单元，低 8 位值存入片内 20H 单元中。

解：如图 11-5 所示，设外部脉冲由  $\overline{\text{INT0}}$  (P3.2) 引脚输入，T0 工作于定时器方式，工作方式 1 (16 位计数)，GATE 设置为 1，TR0 设置为 1，当  $\overline{\text{INT0}}$  为高电平时，启动计数；当  $\overline{\text{INT0}}$  再次变低时，停止计数。此时 T0 中的计数值即为被测正脉冲的宽度。T0 的计数初值设为 0000H。

测试程序如下：

```

MOV    TMOD, #09H           ; T0 定时，方式 1，GATE=1
MOV    TH0,   #00H           ; T0 的计数初值设为 0000H
MOV    TL0,   #00H
MOV    R0,    #20H           ; RAM 的地址指针
LOOP1: JB    P3.2, LOOP1     ; 等待 INT0 变低
        SETB  TR0             ; INT0 变低，启动 T0 准备计数
LOOP2: JNB    P3.2, LOOP2     ; 等待 INT0 变高，启动计数
LOOP3: JB    P3.2, LOOP3     ; 等待 INT0 再次变低
        CLR   TR0             ; INT0 变低即停止计数
MOV    @R0, TL0              ; 存入计数值
INC    R0
MOV    @R0, TH0

```

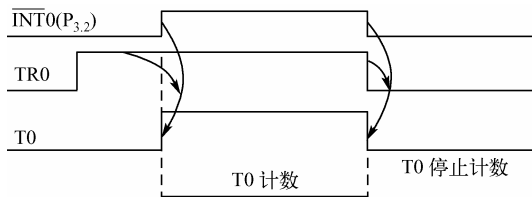


图 11-15 例 11-20 的图

【例 11-21】利用定时/计数器 0 控制产生周期为 1ms 的定时脉冲，由  $\text{P}_{1.0}$  输出。

根据题目要求，输出 1ms 周期的脉冲，定时器的定时时间应为 0.5ms。假定系统时钟频率为 6MHz，T0 选择工作方式 2。

计算计数预置初值： $(2^8 - x) \times 2 \times 10^{-6} = 500 \times 10^{-6}$ ，得  $x = 06$ 。

即定时器 T0 的计数初值为 06H。定时器初始化和中断服务程序如下：

```

ORG    0000H
AJMP   START           ; 转主程序
ORG    000BH           ; T0 的入口地址
AJMP   INT0             ; 转中断服务程序
ORG    0100H
START:  MOV    TMOD, #02H ; 设置 T0 为定时方式 2

```



```

MOV TL0, #06H      ; T0 置初值
MOV TH0, #06H
SETB EA            ; 允许 CPU 中断
SETB ET0           ; 允许 T0 中断
SETB PT0           ; T0 为高优先级中断
SETB TR0           ; 启动 T0
LOOP: LJMP LOOP
ORG 0200H          ; T0 溢出中断服务程序
INT0: CPL P1.0      ; 输出
      RETI          ; 返回主程序

```

【例 11-22】定时器 1 以方式 2 计数，要求每计满 100 次，将 P<sub>1.0</sub> 端取反。

外部计数信号由 T1(P<sub>3.5</sub>)脚引入，每下跳变一次计数器加 1，由程序查询 TF1。方式 2 具有初值自动重装入功能，初始化后不必再置初值。

初值： $x = 2^8 - 100 = 156D = 9CH$ ；TH1 = TL1 = 9CH，TMOD = 60H。

程序如下：

```

MOV TMOD, #60H      ; 设置 T1 为方式 2 计数
MOV TH1, #9CH       ; 赋 T1 初值
MOV TL1, #9CH
SETB TR1            ; 启动 T1
DEL: JBC TF1, REP    ; 查询计数溢出
     AJMP DEL
REP:  CPL P1.0        ; 输出
     AJMP DEL

```

【例 11-23】设计一程序，将 8031（甲）片内 RAM 50H~5FH 中的数据串行发送到 8031（乙）中，并存储于 8031（乙）片内 RAM 40H~4FH 单元中。

假设两单片机的晶振均为 11.0592MHz，波特率为 9600bps，选择串行口方式 3 通信，选择定时器 1 为方式 2 定时。发送与接收程序如下：

单片机（甲）发送程序：

```

TTTT: MOV TMOD, #20H      ; 定时器 1，方式 2 定时
      MOV TL1, #0FDH
      MOV TH1, #0FDH      ; 置定时器初值
      MOV SCON, #0C0H     ; 选择通信方式 3
      SETB TR1            ; 启动定时器 T1
      MOV R0, #50H        ; 首地址 50H→R0
      MOV R7, #10H        ; 数据长度 10H→R7
LOOP:  MOV A, @R0          ; 取数据→A
      MOV C, PSW.0        ;

```

```

                MOV     TB8, C                ; P→TB8
                MOV     SUBF, A                ; 数据→SBUF 启动发送
WAIT:          JBC     TI, CONT                ; 判断发送中断标志
                SJMP    WAIT
CONT:          INC     R0
                DJNZ    R7, LOOP

```

单片机（乙）接收程序：

```

RRRR:          MOV     TMOD, #20H            ; 定时器 1，方式 2 定时
                MOV     TL1, #0FDH
                MOV     TH1, #0FDH            ; 置定时器初值，选定 9600bps
                MOV     SCON, #0D0H          ; 选方式 3，允许接收(REN=1)
                SETB    TR1                    ; 启动定时器 1
                MOV     R0, #40H              ; 首地址 40H→R0
                MOV     R7, #10H              ; 数据长度 10H→R7
LOOP:          JBC     RI, RECE                ; 等待接收
                SJMP    LOOP
RECE:          MOV     A, SBUF
                JB      PSW.0, ONEE            ; 判断接收数据奇偶性
                JB      RB8, ERRR              ; 判断发送端奇偶性
                SJMP    RIGT
ONEE:          JNB     RB8, ERRR
RIGT:          MOV     @R0, A                  ; 接收正确
                INC     R0
                DJNZ    R7, LOOP
ERRR:          ;                               ; 接收错误

```

## 习题

1. MCS-51 系列单片机由哪些基本部件组成？
2. MCS-51 单片机共有几个工作寄存器组？如何选择？
3. MCS-51 的  $\overline{\text{EA}}$  信号有何功能？在使用 8031 时  $\overline{\text{EA}}$  信号引脚如何处理？
4. MCS-51 系列单片机有几个中断源？各中断标志是如何产生的？
5. MCS-51 系列单片机有几个外部中断和内部中断？
6. MCS-51 系列单片机的中断系统中有几个优先级？
7. MCS-51 系列单片机的中断矢量地址分别是多少？
8. 8051 单片机内部设有几个定时/计数器？
9. MCS-51 系列单片机的定时/计数器有哪几种工作方式？各种工作方式的特点是什么？如何选择和设定定时器的工作方式？

10. MCS-51 系列单片机中的定时器有哪几个专用寄存器？它们各自的作用是什么？
11. 怎样计算定时器计数的初值？
12. 编写一个定时间隔为 25ms 的程序，晶振频率为 6MHz。
13. 8051 定时器做定时和计数时其计数脉冲分别由谁提供？
14. 什么是指令系统？MCS-51 指令系统具有几种寻址方式？
15. 试述 MOVX 和 MOVC 指令的异同之处。
16. 编一数据块搬迁程序。将外部 RAM 2000H~202FH 单元中的内容，移入内部 RAM 20H~4FH 单元中。
17. 请将片外数据存储器地址为 2000H~2080H 的数据块，全部移到 2800H~2880H 中，并将原数据块区域全部清 0。

# 附录A 8088/8086 运算指令对标志位的影响

指    令	状  态  标  志						指    令	状  态  标  志					
	O	C	A	S	Z	P		O	C	A	S	Z	P
(1) 加法和减法	↑ ↑ ↑ ↑ ↑	↑	↑	↑	↑	↑	XOR, TEST	0	0	×	↑	↑	↑
ADD, ADC		↑	↑	↑	↑	↑	(5) 移位和循环	↑	↑	×	↑	↑	↑
SUB, SBB		↑	↑	↑	↑	↑	SHL, SHR (一次)	↑	×	×	×	↑	↑
CMP, NEG		↑	↑	↑	↑	↑	SHL, SHR (CL)	↑	×	↑	×	↑	↑
CMPS, SCAS		↑	↑	↑	↑	↑	SAR	0	↑	×	↑	↑	↑
INC, DEC	↑	•	↑	↑	↑	↑	ROL, ROR (一次)	↑	↑	•	•	•	•
(2) 乘法和除法	↑ ×	↑	×	×	×	×	ROL, ROR (CL)	×	↑	•	•	•	•
MUL, IMUL		↑	×	×	×	×	RCL, RCR (一次)	↑	↑	•	•	•	•
DIV, IDIV		×	×	×	×	×	RCL, RCR (CL)	×	↑	•	•	•	•
(3) 十进制运算	×	↑	↑	↑	↑	↑	(6) 标志操作	↑	↑	↑	↑	↑	↑
DAA, DAS		×	×	×	×	×	POPF, IRET	•	↑	↑	↑	↑	↑
AAA, AAS		×	×	×	×	×	SAHF	•	1	•	•	•	•
AAM, AAD	×	×	×	×	×	×	STC	•	0	•	•	•	•
(4) 逻辑运算	0	0	×	×	×	×	CLC	•	0	•	•	•	•
AND, OR		0	0	×	×	×	CMC	•	1	•	•	•	•

符号说明:

↑: 有影响      0: 把标志置为 0      •: 无影响 (维持原来状态)      1: 把标志置为 1      ×: 无定义 (不定)       $\bar{C}$ : C 标志取反

## 附录B DOS功能调用（INT 21H）

AH	功 能	调 用 参 数	返 回 参 数
00	程序中止 (同 INT 20H)	CS=程序段前缀	
01	键盘输入并回显		AL=输入字符
02	显示输出	DL=输出字符	
03	异步通信输入		AL=输入数据
04	异步通信输出	DL=输出数据	
05	打印机输出	DL=输出字符	
06	直接控制台 I/O	DL=FF (输入) DL=字符 (输出)	AL=输入字符
07	键盘输入 (无回显)		AL=输入字符
08	键盘输入 (无回显) 检测 Ctrl-Break		AL=输入字符
09	显示字符串	DS: DX=串地址 \$' 结束字符串	
0A	键盘输入到缓冲区	DS: DX=缓冲区首地址 (DS: DX)=缓冲区最大字符数	(DS: DX+1)=实际输入的字符数
0B	检验键盘状态		AL=00, 无输入 AL=0FF, 有输入
0C	清除输入缓冲区并请求指定的输入功能	AL=输入功能号 (1, 6, 7, 8, A)	
0D	磁盘复位		清除文件缓冲区
0E	指定当前缺省的磁盘驱动器	DL=驱动器号 0=A, 1=B, ...	AL=驱动器数
0F	打开文件	DS: DX=FCB 首地址	AL=00, 文件找到 AL=0FF, 文件未找到
10	关闭文件	DS: DX=FCB 首地址	AL=00, 目录修改成功 AL=0FF, 目录未找到文件
11	查找第一个目录项	DS: DX=FCB 首地址	AL=00, 找到 AL=0FF, 未找到
12	查找下一个目录项	DS: DX=FCB 首地址 (文件名中带*或?)	AL=00, 找到 AL=0FF, 未找到
13	删除文件	DS: DX=FCB 首地址	AL=00, 删除成功 AL=0FF, 未找到

续表

AH	功 能	调 用 参 数	返 回 参 数
14	顺序读	DS: DX=FCB 首地址	AL = 00, 读成功 AL= 01, 文件结束,记录中无数据 AL= 02 , DATA 空间不够 AL= 03, 文件结束, 记录不完整
15	顺序写	DS: DX=FCB 首地址	AL = 00, 写成功 AL= 01, 盘满 AL= 02, DATA 空间不够
16	建文件	DS: DX=FCB 首地址	AL = 00, 建立成功 AL= 0FF, 无磁盘空间
17	文件改名	DS: DX=FCB 首地址 (DS: DX+1) =旧文件名 (DS: DX+17) =新文件名	AL = 00, 建立成功 AL= 0FF, 未成功
19	取当前缺省磁盘驱动器		AL=缺省的驱动器号 0=A, 1=B, 2=C, ...
1A	置 DATA 地址	DS: DX=DATA 地址	
1B	取缺省驱动器 FAT 信息		AL=每簇的扇区数 DS: BX=FTA 标识字节 CS=物理扇区的大小 DX=缺省驱动器的簇数
1C	取任一驱动器 FAT 信息	DL=驱动器号	同上
21	随机读	DS: DX=FCB 首地址	AL = 00, 读成功 AL= 01, 文件结束 AL= 02, 缓冲区溢出 AL= 03, 缓冲区不满
22	随机写	DS: DX=FCB 首地址	AL = 00, 写成功 AL= 01, 盘满 AL= 02, 缓冲区溢出
23	测定文件大小	DS: DX=FCB 首地址	AL=00, 成功 文件长度填入 FCB
24	设置随机记录号	DS: DX=FCB 首地址	AL=0FF, 未找到
25	设置中断向量	DS: DX=中断向量 AL=中断类型号	
26	建立程序段前缀	DX=新的程序段的段前缀	

续表

AH	功 能	调 用 参 数	返 回 参 数
27	随机分块读	DS: DX=FCB 首地址 CX=记录数	AL = 00, 读成功 AL = 01, 文件结束 AL = 02, 缓冲区太小, 传输结束 AL = 03, 缓冲区不满
28	随机分块写	DS: DX=FCB 首地址 CX=记录数	AL = 00, 写成功 AL = 01, 盘满 AL = 02, 缓冲区溢出
29	分析文件名	ES: DI=FCB 首地址 DS: SI=ASCIIZ 串 AL=控制分析标志	AL = 00, 标准文件 AL = 01, 多义文件 AL = 02, 非法盘符
2A	取日期		CX=年 DH: DL=月: 日 (二进制)
2B	设置日期	CX: DH: DL=年: 月: 日	AL = 00, 成功; AL = 0FF 无效 CH: CL=时: 分
2C	取时间		DH: DL=秒: 1/100 秒 AL = 00, 成功
2D	设置时间	CH: CL=时: 分 DH: DL=秒: 1/100 秒	AL = FF, 无效
2E	置磁盘自动 读/写标志	AL=00, 关闭标志 AL=01, 打开标志	ES: BX=缓冲区首址 AH=发行号, AL=版号
2F	取磁盘缓冲区的首址		
30	取 DOS 版本号		
31	结束并驻留	AL=返回码, DX=驻留区大小	
33	Ctrl-Break 检测	DL=00, 取状态 AL=01, 置状态 (DL) DL=00, 关闭检测 DL=01, 打开检测	DL = 00, 关 Ctrl-Break 检测 DL = 01, 打开 Ctrl-Break 检测
35	取中断向量	AL=中断类型	ES: BX=中断向量
36	取空闲磁盘空间	DL=驱动器号 0=缺省, 1=A, 2=B, ...	成功: AX=每簇扇区数 BX=有效簇数 CX=每扇区字节数 DX=总簇数 失败: AX=FFFF
38	置/取国家信息	DS: DX=信息区首地址	BX=国家码 (国际电话前缀码) AX=错误码

续表

AH	功 能	调 用 参 数	返 回 参 数
39	建立子目录 (MKDIR)	DS: DX=ASCII 串地址	AX=错误码
3A	删除子目录 (RMDIR)	DS: DX=ASCII 串地址	AX=错误码
3B	改变当前目录 (CHDIR)	DS: DX=ASCII 串地址	AX=错误码
3C	建立文件	DS: DX=ASCII 串地址 CX=文件属性	成功: AX=文件代号 失败: AX=错误码
3D	打开文件	DS: DX=ASCII 串地址 AL=0, 读 AL=1, 写 AL=2, 读/写	成功: AX=文件代号 失败: AX=错误码
3E	关闭文件	BX=文件号	失败: AX=错误码
3F	读文件或设备	DS: DX=数据缓冲区地址 BX=文件代号 CX=读取的字节数	读成功: AX=实际读入的字节数 AX=0 已到文件尾 读出错: AX=错误码
40	写文件或设备	DS: DX=数据缓冲区地址 BX=文件代号 CX=写入的字节数	写成功: AX=实际读入的字节数 写出错: AX=错误码
41	删除文件	DS: DX=ASCII 串地址	成功: AX=00 出错: AX=错误码 (2, 5)
42	移动文件指针	BX=文件代号 CX: DX=位移量 AL=移动方式 (0, 1, 2)	成功: DX: AX=新指针位置 出错: AX=错误码
43	置/取文件属性	DS: DX=ASCII 串地址 AL=0, 取文件属性 AL=1, 置文件属性 CX=文件属性	成功: CX=文件属性 失败: AX=错误码
44	设备文件 I/O 控制	BX=文件代号 AL=0, 取状态 AL=1, 置状态 DX AL=2, 读数据 AL=3, 写数据 AL=6, 取输入状态 AL=7, 取输出状态	DX=设备信息
45	复制文件代号	BX=文件代号 1	成功: AX=文件代号 2 失败: AX=错误码



续表

AH	功 能	调 用 参 数	返 回 参 数
46	人工复制文件代号	BX=文件代号 1 CX=文件代号 2	失败: AX=错误码
47	取当前目录路径名	DL=驱动器号 DS: SI=ASCIIZ 串地址	(DS: SI)=ASCIIZ 串 失败: AX=错误码
48	分配内存空间	BX=申请内存容量	成功: AX=分配内存首址 失败: BX=最大可用空间
49	释放内存空间	ES=内存起始段地址	失败: AX=错误码
4A	高速已分配的存储块	ES=原内存起始地址 BX=再申请的容量	失败: BX=最大可用空间 AX=错误码
4B	装配/执行程序	DS: DX=ASCIIZ 串地址 ES: BX=参数区首地址 AL=0, 装入执行 AL=3, 装入不执行	失败: AX=错误码
4C	带返回码结束	AL=返回码	
4D	取返回代码		AX=返回代号
4E	查找第一个匹配文件	DS: DX=ASCIIZ 串地址 CX=属性	AX=出错代码 (02, 18)
4F	查找下一个匹配文件	DS: DX=ASCIIZ 串地址 (文件名中带? 或*)	AX=出错代码 (18)
54	取盘自动读/写标志		AL=当前标志值
56	文件改名	DS: DX=ASCIIZ 串 (旧) ES: DI=ASCIIZ 串 (新)	AX=出错码 (03, 05, 17)
57	置/取文件日期和时间	BX=文件代号 AL=0, 读取 AL=1, 设置 (DX: CX)	DX: CX=日期和时间 失败: AX=错误码
58	取/置分配策略码	AL=0, 取码 AL=1, 置码 (BX) BX=策略码	成功: AX=策略码 失败: AX=错误码
59	取扩充错误码		AX=扩充错误码 BH=错误类型 BL=建议的操作
5A	建立临时文件	CX=文件属性 DS: DX=ASCIIZ 串地址	CH=错误场所 成功: AX=文件代号 失败: AX=错误码

续表

AH	功    能	调  用  参  数	返  回  参  数
5B	建立新文件	CX=文件属性	成功：AX=文件代号
		DS: DX=ASCIIZ 串地址	失败：AX=错误码
5C	控制文件存取	AL = 00, 封锁	失败：AX=错误码
		AL= 01, 开启	
		BX=文件代号	
		CX: DX=文件位移	
		SI: DI=文件长度	
62	取程序段前缀地址		BX=PSP 地址

\*    AH=0～2E，适用 DOS1.0 以上版本；AH=2F～57，适用 DOS2.0 以上版本；AH=58～62，适用 DOS3.0 以上版本。

# 附录C BIOS功能调用

INT	AH	功 能	调 用 参 数	返 回 参 数
10	0	设置显示方式	AL=00, 40×25 黑白方式 AL=01, 40×25 彩色方式 AL=02, 80×25 黑白方式 AL=03, 80×25 彩色方式 AL=04, 320×200 彩色图形方式 AL=05, 320×200 黑白图形方式 AL=06, 640×200 黑白图形方式 AL=07, 80×25 单色文本方式 AL=08, 160×200 16 色图形(PCjr) AL=09, 320×200 16 色图形(PCjr) AL=0A, 640×200 16 色图形(PCjr) AL=0B, 保留(EGA) AL=0C, 保留(EGA) AL=0D, 320×200 彩色图形(EGA) AL=0E, 640×200 彩色图形(EGA) AL=0F, 640×350 黑白图形(EGA) AL=10, 640×350 彩色图形(EGA) AL=11, 640×480 单色图形(EGA) AL=12, 640×480 16 色图形(EGA) AL=13, 320×320 256 色图形(EGA) AL=40, 80×30 彩色文本(CGE400) AL=41, 80×50 彩色文本(CGE400) AL=42, 640×400 彩色文本(CGE400)	
10	1	置光标类型	(CH) <sub>0~3</sub> =光标起始行 (CL) <sub>0~3</sub> =光标结束行	
10	2	置光标位置	BH=页号 DH, DL=行, 列	
10	3	读光标位置	BH=页号	CH=光标起始行 DH, DL=行, 列
10	4	读光笔位置		AH=0 光笔未触发 AH=1 光笔触发

续表

INT	AH	功 能	调 用 参 数	返 回 参 数
10	5	置显示页	AL=页号	CH=像素行 BX=像素列 DH=字符行 DL=字符列
10	6	屏幕初始化或上卷	AL=上卷行号 AL=0, 整个窗口空白 BH=卷入行属性 CH=左上角行号 CL=左上角列号 DH=右下角行号 DL=右下角列号	
10	7	屏幕初始化或下卷	AL=下卷行数 AL=0, 整个窗口空白 BH=卷入行属性 CH=左上角行号 CL=左上角列号 DH=右下角行号 DL=右下角列号	
10	8	读光标位置的字符和属性	BH=显示页	
10	9	在光标位置显示 字符及其属性	BH=显示页, AL=字符 BL=属性, CX=字符重复次数	AH=属性, AL=字符
10	A	在光标位置显示字符	BH=显示页, AL=字符 CX=字符重复次数	
10	B	置彩色调板 (320×200 图形)	BH=彩色调板 ID BL=和 ID 配套使用的颜色	
10	C	写像素	DX=行(0~199), CX=行(0~639) AL=像素值	
10	D	读像素	DX=行(0~199), CX=行(0~639)	AL=像素值

续表

INT	AH	功 能	调 用 参 数	返 回 参 数
10	E	显示字符(光标前移)	AL=字符, BL=前景色	AH=字符列数, AL=显示方式
10	F	取当前显示方式		
10	13	显示字符串(适用 AT)	ES: BP=串地址 CX=串长度 DH, DL=起始行, 列 BH=页号 AL=0, BL=属性 串: char, char, ... AL=1, BL=属性 串: char, char, ... AL=2 串: char, attr, char, attr, ... AL=3 串: char, attr, char, attr, ...	
11		设备检验		
12		测定存储器容量		AX=返回值 bit0=1, 配有磁盘 bit1=1,80287 协处理器 bit4,5=01,40×25BW (彩色板) bit4,5=10,80×25BW(彩色板) bit4,5=11,80×25BW(黑白板) bit6,7=软盘驱动器号 bit9,10,11=RS-232 板号 bit12=游戏适配器 bit13=串行打印机 bit14,15=打印机号 AX=字节数(KB)
13	0	软盘系统复位		AL=状态字节 读成功: AH=0 AL=读取的扇区数 读失败: AH=出错代码
13	1	读磁盘状态		
13	2	读磁盘	AL=扇区数 CH, CL=磁道号, 扇区号 DH, DL=磁头号, 驱动器号 ES:BX=数据缓冲区地址	

续表

INT	AH	功 能	调 用 参 数	返 回 参 数
13	3	写磁盘	同上	写成功: AH=0 AL=写入的扇区数 写失败: AH=出错代码
13	4	检验磁盘扇区	同上(ES: BX 不设置)	成功: AH=0 AL=检验的扇区数 失败: AH=出错代码
13	5	格式化盘磁道	ES: BX=磁道地址	成功: AH=0 失败: AH=出错代码
14	0	初始化串行通信口	AL=初始化参数 DX=通信口号(0, 1)	AH=通信口状态 AL=调制解调器状态
14	1	向串行通信口写字符	AL=字符 DX=通信口号(0, 1)	写成功: (AH) <sub>7</sub> =0 写失败: (AH) <sub>7</sub> =1 (AH) <sub>0~6</sub> =通信口状态
14	2	从串行通信口读字符	DX=通信口号(0, 1)	读成功: (AH) <sub>7</sub> =0 (AL)=字符 读失败: (AH) <sub>7</sub> =1 (AH) <sub>0~6</sub> =通信口状态
14	3	取通信口状态	DX=通信口号(0, 1)	AH=通信口状态 AL=调制解调器状态
15	0	启动盒式磁带马达	ES: BX=数据传输区地址 CX=字节数	AH=状态字节 AL=00, 读成功 AL=01, 冗余检验错 AL=02, 无数据传输 AL=04, 无引导 AL=80, 非法命令
15	1	停止盒式磁带马达		
15	2	磁带分块读		
15	3	磁带分块写	DS: BX=数据传输区地址 CX=字节数	AH=状态字节 (同上)
16	0	从键盘读字符		AL=字符码 AH=扫描码
16	1	读键盘缓冲区字符		ZF=0 AH=字符码 AL=扫描码 ZF=1 缓冲区空

续表

INT	AH	功 能	调 用 参 数	返 回 参 数
16	2	取键盘状态字节		AL=键盘状态字节
17	0	打印字符	AL=字符	AH=打印机状态字节
		回送状态字节	DX=打印机号	
17	1	初始化打印机	DX=打印机号	AH=打印机状态字节
		回送状态字节		
17	2	取状态字节	DX=打印机号	AH=打印机状态字节
1A	0	读时钟		CH: CL=时: 分 DH: DL=秒: (1/100)秒
1A	1	置时钟	CH: CL=时: 分 DH: DL=秒: (1/100)秒	
1A	2	读时钟 (适用 AT)		CH: CL=时: 分(BCD) DH: DL=秒: (1/100)秒(BCD)
1A	6	置报警时间 (适用 AT)	CH: CL=时: 分(BCD) DH: DL=秒: (1/100)秒(BCD)	
1A	7	消除报警 (适用 AT)		

## 附录D MCS-51 指令表

MCS-51 指令系统所用符号和含义：

addr11	11 位地址
addr16	16 位地址
bit	位地址
rel	相对偏移量，为 8 位有符号数（补码形式）
direct	直接地址单元
#data	立即数
Rn	工作寄存器 R0~R7
A	累加器
Ri	i=0,1,R0 或 R1
X	片内 RAM 中的直接地址或寄存器
@	表示间址寄存器的符号
(X)	表示直接地址 X 中的内容；在间接寻址方式中，表示间址寄存器地址 X 指出的地址单元中的内容
→	数据传送方向
∧	逻辑与
∨	逻辑或
⊕	逻辑异或
✓	对标志位产生影响
×	不影响标志位

十六进制 制代码	助 记 符	功 能	对标志位影响				字节 数	周期 数	
			P	OV	AC	CY			
算术运算指令									
28~2F	ADD A,Rn	A+Rn→A	✓	✓	✓	✓	1	1	
25	ADD A,direct	A+(direct) →A	✓	✓	✓	✓	2	1	
26,27	ADD A,@Ri	A+(Ri) →A	✓	✓	✓	✓	1	1	
24	ADD A,#data	A+data→A	✓	✓	✓	✓	2	1	
38~3F	ADDC A,Rn	A+Rn+CY→A	✓	✓	✓	✓	1	1	
35	ADDC A,direct	A+(direct)+CY→A	✓	✓	✓	✓	2	1	
36,37	ADDC A,@Ri	A+(Ri)+CY→A	✓	✓	✓	✓	1	1	
34	ADDC A,#data	A+data+CY→A	✓	✓	✓	✓	2	1	
98~9F	SUBB A,Rn	A-Rn-CY→A	✓	✓	✓	✓	1	1	
95	SUBB A,direct	A- (direct)-CY→A	✓	✓	✓	✓	2	1	
96,97	SUBB A,@Ri	A- (Ri)→CY→A	✓	✓	✓	✓	1	1	
94	SUBB A,#data	A-data-CY→A	✓	✓	✓	✓	2	1	
04	INC A	A+1→A	✓	×	×	×	1	1	



续表

十六进制代码	助记符	功能	对标志位影响				字节数	周期数
			P	OV	AC	CY		
08~0F	INC Rn	$Rn+1 \rightarrow Rn$		×	×	×	1	
05	INC direct	$(direct)+1 \rightarrow (direct)$	×	×	×	×	2	1
06,07	INC @Ri	$(Ri)+1 \rightarrow (Ri)$	×	×	×	×	1	1
A3	INC DPTR	$DPTR+1 \rightarrow DPTR$	×				1	1
14	DEC A	$A-1 \rightarrow A$		×	×	×	1	2
18~1F	DEC Rn	$Rn-1 \rightarrow Rn$	√	×	×	×	1	1
			×					1
15	DEC direct	$(direct)-1 \rightarrow (direct)$	×	×	×	×	2	1
16,17	DEC @Ri	$(Ri)-1 \rightarrow (Ri)$	×	×	×	×	1	1
A4	MUL AB	$A \cdot B \rightarrow AB$	√	√	×	0	1	4
84	DIV AB	$A/B \rightarrow AB$	√	√	×	0	1	4
D4	DAA	对 A 进行十进制调整	√	×	√	√	1	1
逻辑运算指令								
58~5F	ANL A,Rn	$A \wedge Rn \rightarrow A$	√	×	×	×	1	1
55	ANL A,direct	$A \wedge (direct) \rightarrow A$	√	×	×	×	2	1
56,57	ANL A,@Ri	$A \wedge (Ri) \rightarrow A$	√	×	×	×	1	1
54	ANL A,#data	$A \wedge data \rightarrow A$	√	×	×	×	2	1
52	ANL direct,A	$(direct) \wedge A \rightarrow A$	×	×	×	×	2	1
53	ANL direct,#data	$(direct) \wedge data \rightarrow A$	×	×	×	×	3	2
48~4F	ORL A,Rn	$A \vee Rn \rightarrow A$	√	×	×	×	1	1
45	ORL A,direct	$A \vee (direct) \rightarrow A$	√	×	×	×	2	1
46,47	ORL A,@Ri	$A \vee (Ri) \rightarrow A$	√	×	×	×	1	1
44	ORL A,#data	$A \vee data \rightarrow A$	√	×	×	×	2	1
42	ORL direct,A	$(direct) \vee A \rightarrow (direct)$	×	×	×	×	2	1
43	ORL direct,#data	$(direct) \vee data \rightarrow (direct)$	×	×	×	×	3	2
68~6F	XRL A,Rn	$A \oplus Rn \rightarrow A$	√	×	×	×	1	1
65	XRL A,direct	$A \oplus (direct) \rightarrow A$	√	×	×	×	2	1
66,67	XRL A,@Ri	$A \oplus (Ri) \rightarrow A$	√	×	×	×	1	1
64	XRL A,#data	$A \oplus data \rightarrow A$	√	×	×	×	2	1
62	XRL direct,A	$(direct) \oplus A \rightarrow (direct)$	×	×	×	×	2	1
63	XRL direct,#data	$(direct) \oplus data \rightarrow (direct)$	×	×	×	×	3	2
E4	CLR A	$0 \rightarrow A$	√	×	×	×	1	1
F4	CPL A	$\overline{A} \rightarrow A$	×	×	×	×	1	1
23	RL A	A 循环左移一位	×	×	×	×	1	1
33	RLC A	A 带进位循环左移一位	√	×	×	√	1	1
03	RR A	A 循环右移一位	×	×	×	×	1	1
13	RRC A	A 带进位循环右移一位	√	×	×	√	1	1
C4	SWAP A	A 半字节交换	×	×	×	×	1	1

续表

十六进制 制代码	助 记 符	功 能	对标志位影响				字节 数	周期 数
			P	OV	AC	CY		
数据传送指令								
E8~EF	MOV A,Rn	Rn→A	√	×	×	×	1	1
E5	MOV A,direct	(direct)→A	√	×	×	×	2	1
E6,E7	MOV A,@Ri	(Ri)→A	√	×	×	×	1	1
74	MOV A,#data	data→A	√	×	×	×	2	1
F8~FF	MOV Rn,A	A→Rn	×	×	×	×	1	1
A8~AF	MOV Rn,direct	(direct)→Rn	×	×	×	×	2	2
78~7F	MOV Rn,#data	data→Rn	×	×	×	×	2	1
F5	MOV direct,A	A→(direct)	×	×	×	×	2	1
88~8F	MOV direct,Rn	Rn→(direct)	×	×	×	×	2	2
85	MOV direct1,direct2	(direct2)→(direct1)	×	×	×	×	3	2
86,87	MOV direct,@Ri	(Ri)→(direct)	×	×	×	×	2	2
75	MOV direct,#data	data→(direct)	×	×	×	×	3	2
F6,F7	MOV @Ri,A	A→(Ri)	×	×	×	×	1	1
A6,A7	MOV @Ri,direct	(direct)→(Ri)	×	×	×	×	2	2
76,77	MOV @Ri,#data	data→(Ri)	×	×	×	×	2	1
90	MOV DPTR,#data16	data16→DPTR	×	×	×	×	3	2
93	MOVC A,@A+DPTR	(A+DPTR)→A	√	×	×	×	1	2
83	MOVC A,@A+PC	PC+1→PC,(A+PC)→A	√	×	×	×	1	2
E2,E3	MOVX A,@Ri	(Ri)→A	√	×	×	×	1	2
E0	MOVX A,@DPTR	(DPTR)→A	√	×	×	×	1	2
F2,F3	MOVX @Ri,A	A→(Ri)	×	×	×	×	1	2
F0	MOVX @DPTR,A	A→(DPTR)	×	×	×	×	1	2
C0	PUSH direct	SP+1→SP,(direct)→(SP)	×	×	×	×	2	2
D0	POP direct	(SP)→(direct),SP-1→SP	×	×	×	×	2	2
C8~CF	XCH A,Rn	A↔Rn	√	×	×	×	1	1
C5	XCH A,direct	A↔(direct)	√	×	×	×	2	1
C6,C7	XCH A,@Ri	A↔(Ri)	√	×	×	×	1	1
D6,D7	XCHD A,@Ri	A0~3↔(Ri)0~3	√	×	×	×	1	1
位操作指令								
C3	CLR C	0→CY	×	×	×	√	1	1
C2	CLR bit	0→bit	×	×	×		2	1
D3	SETB C	1→CY	×	×	×	√	1	1
D2	SETB bit	1→bit	×	×	×		2	1
B3	CPL C	$\overline{\text{CY}} \rightarrow \text{CY}$	×	×	×	√	1	1

续表

十六进制代码	助记符	功 能	对标志位影响				字节数	周期数
			P	OV	AC	CY		
B2	CPL bit	$\overline{\text{bit}} \rightarrow \text{bit}$	×	×	×		2	1
82	ANL C,bit	$\text{CY} \wedge \text{bit} \rightarrow \text{CY}$	×	×	×	✓	2	2
B0	ANL C,/bit	$\text{CY} \wedge \overline{\text{bit}} \rightarrow \text{CY}$	×	×	×	✓	2	2
72	ORL C,bit	$\text{CY} \vee \text{bit} \rightarrow \text{CY}$	×	×	×	✓	2	2
A0	ORL C,/bit	$\text{CY} \vee \overline{\text{bit}} \rightarrow \text{CY}$	×	×	×	✓	2	2
A2	MOV C,bit	$\text{bit} \rightarrow \text{CY}$	×	×	×	✓	2	1
92	MOV bit,C	$\text{CY} \rightarrow \text{bit}$	×	×	×	×	2	2
控制转移指令								
*1	ACALL addr 11	$\text{PC}+2 \rightarrow \text{PC}, \text{SP}+1 \rightarrow \text{SP}, \text{PCL} \rightarrow (\text{SP})$ $\text{SP}+1 \rightarrow \text{SP}, \text{PCH} \rightarrow (\text{SP}), \text{Addr11} \rightarrow \text{PC10} \sim 0$	×	×	×	×	2	2
12	LCALL addr 16	$\text{PC}+3 \rightarrow \text{PC}, \text{SP}+1 \rightarrow \text{SP}, \text{PCL} \rightarrow (\text{SP}),$ $\text{SP}+1 \rightarrow \text{SP}, \text{PCH} \rightarrow (\text{SP}), \text{addr16} \rightarrow \text{PC}$	×	×	×	×	3	2
22	RET	$(\text{SP}) \rightarrow \text{PCH}, \text{SP}-1 \rightarrow \text{SP}, (\text{SP}) \rightarrow \text{PCL}$ $\text{SP}-1 \rightarrow \text{SP}$	×	×	×	×	1	2
32	RETI	$(\text{SP}) \rightarrow \text{PCH}, \text{SP}-1 \rightarrow \text{SP}, (\text{SP}) \rightarrow \text{PCL}$ $\text{SP}-1 \rightarrow \text{SP}$	×	×	×	×	1	2
*1	AJMP addr 11	$\text{PC}+2 \rightarrow \text{PC}, \text{addr11} \rightarrow \text{PC10} \sim 0$	×	×	×	×	2	2
02	LJMP addr 16	$\text{Addr16} \rightarrow \text{PC}$	×	×	×	×	3	2
80	SJMP rel	$\text{PC}+2 \rightarrow \text{PC}, \text{PC}+\text{rel} \rightarrow \text{PC}$	×	×	×	×	2	2
73	JMP @A+DPTR	$(\text{A}+\text{DPTR}) \rightarrow \text{PC}$	×	×	×	×	1	2
60	JZ rel	$\text{PC}+2 \rightarrow \text{PC}$ , 若 $\text{A}=0, \text{PC}+\text{rel} \rightarrow \text{PC}$	×	×	×	×	2	2
70	JNZ rel	$\text{PC}+2 \rightarrow \text{PC}$ , 若 $\text{A}$ 不等于 0, 则 $\text{PC}+\text{rel} \rightarrow \text{PC}$	×	×	×	×	2	2
40	JC rel	$\text{PC}+2 \rightarrow \text{PC}$ , 若 $\text{CY}=1$ , 则 $\text{PC}+\text{rel} \rightarrow \text{PC}$	×	×	×	×	2	2
50	JNC rel	$\text{PC}+2 \rightarrow \text{PC}$ , 若 $\text{CY}=0$ , 则 $\text{PC}+\text{rel} \rightarrow \text{PC}$	×	×	×	×	2	2
20	JB bit,rel	$\text{PC}+3 \rightarrow \text{PC}$ , 若 $\text{bit}=0$ , 则 $\text{PC}+\text{rel} \rightarrow \text{PC}$	×	×	×	×	3	2
30	JNB bit,rel	$\text{PC}+3 \rightarrow \text{PC}$ , 若 $\text{bit}=1$ , 则 $\text{PC}+\text{rel} \rightarrow \text{PC}$	×	×	×	×	3	2
10	JBC bit,rel	$\text{PC}+3 \rightarrow \text{PC}$ , 若 $\text{bit}=1$ , 则 $0 \rightarrow \text{bit}$ $\text{PC}+\text{rel} \rightarrow \text{PC}$				3	2	
B5	CJNE A,direct,rel	$\text{PC}+3 \rightarrow \text{PC}$ , 若 $\text{A}$ 不等于 (direct), 则 $\text{PC}+\text{rel} \rightarrow \text{PC}$ ; 若 $\text{A} < (\text{direct})$ , 则 $1 \rightarrow \text{CY}$	×	×	×	✓	3	2

续表

十六进制 制代码	助 记 符	功 能	对标志位影响				字节 数	周期 数
			P	OV	AC	CY		
B4	CJNE A,#data,rel	PC+3→PC,若 A 不等于 data,则 PC+rel→PC; 若 A 小于 data,则 1→CY	×	×	×	√	3	2
B8~BF	CJNE Rn,#data,rel	PC+3→PC,若 Rn 不等于 data,则 PC+rel→PC; 若 Rn 小于 data,则 1→CY	×	×	×	√	3	2
B6~B7	CJNE @Ri,#data rel	PC+3→PC,若 Ri 不等于 data,则 PC+rel→PC; 若 Ri 小于 data,则 1→CY	×	×	×	√	3	2
D8~DF	DJNZ Rn,rel	Rn-1→Rn,PC+2→,若 Rn 不等于 0, 则 PC+rel→PC	×	×	×	×	2	2
D5	DJNZ direct,rel	PC+2→PC,(direct)-1→(direct) 若(direct)不等于 0 则 PC+rel→PC	×	×	×	×	3	2
00	NOP	空操作	×	×	×	×	1	1

## 参 考 文 献

- 1 艾德才, 姚嘉康, 龚涛. 微机接口技术实用教程. 北京: 清华大学出版社, 2003
- 2 郑学坚, 周斌. 微型计算机原理及应用. 北京: 清华大学出版社, 2003
- 3 周明德. 微型计算机系统原理及应用. 北京: 清华大学出版社, 2000
- 4 张菊鹏等. 计算机硬件技术基础. 北京: 清华大学出版社, 2001
- 5 艾德才等. 计算机硬件技术基础. 北京: 中国水利水电出版社, 2003
- 6 李伯成. 微型计算机原理及接口技术. 北京: 电子工业出版社, 2002
- 7 徐大诚, 邹丽新, 丁建强. 微型计算机控制技术的应用. 北京: 高等教育出版社, 2003
- 8 戴梅萼. 微型计算机技术及应用. 北京: 清华大学出版社, 1993
- 9 董方武等. 微机接口技术. 北京: 中国水利水电出版社, 2001
- 10 李芷等. 微机原理与接口技术. 北京: 电子工业出版社, 2003
- 11 邹逢兴. 计算机硬件技术及应用基础. 长沙: 国防科技大学出版社, 2001
- 12 胡瑞雯. 智能检测与控制系统. 西安: 西安交通大学出版社, 1991
- 13 俸远祯等. 计算机组成原理与汇编语言程序设计. 北京: 电子工业出版社, 1997
- 14 赵晓安. MCS-51 单片机原理及应用. 天津: 天津大学出版社, 2001
- 15 张迎新. 单片微型计算机原理、应用及接口技术. 北京: 国防工业出版社, 2002
- 16 赵新民. 智能仪器设计基础. 哈尔滨: 哈尔滨工业大学出版社, 1999
- 17 翟生辉, 冯毛官. 单片计算机原理与应用. 西安: 西安交通大学出版社, 2000
- 18 曹琳琳, 曹巧媛. 单片机原理及接口技术. 长沙: 国防科技大学, 2000
- 19 何立民. MCS-51 系列单片机应用系统设计. 北京: 北京航空航天大学出版社, 1994